GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO FINAL DE GRADO

Diseño de una red de monitorización del entorno basada en Arduino, Raspberry Pi y XBee

Autor:
Gabriel SANTAMARÍA BOMBOI

Supervisor:
Rafael AMORÓS AMIGUET
Tutor académico:
Juan Carlos FERNÁNDEZ FERNÁNDEZ

Fecha de lectura: 28 de octubre de 2016
Curso académico 2015/2016

Resumen

En esta memoria se va a proceder a describir el diseño y proceso de desarrollo del prototipo de una red de monitorización del entorno orientada principalmente a ambientes urbanos (las denominadas *smart-cities* o ciudades inteligentes) y agrícolas (cultivos inteligentes). Este documento recoge los detalles del proceso de diseño y las decisiones tomadas, así como todos aquellos aspectos técnicos que se han tenido en cuenta para la obtención del producto final. El diseño de esta red de monitorización incluye el uso de Arduinos como nodos de la red, que se encargan de recoger datos del entorno a través de los sensores de los que disponen, y de una Raspberry Pi como coordinador y puerta de enlace, de modo que ejercerá de intermediario entre los nodos y el servidor al que se envían los datos, que se almacenarán en una base de datos MongoDB. Como medio de comunicación entre los nodos y el coordinador de la red se dispondrá de módulos inalámbricos XBee serie 1 Pro. Mientras que para la transmisión de los datos a la red y al servidor la Raspberry empleará una conexión WiFi o física en función de los requerimientos de la instalación. Además, el proyecto incluye el desarrollo de una plataforma web básica para la consulta de los datos recogidos y la gestión de nodos, sensores, alertas y usuarios.

Palabras clave

Arduino UNO, Raspberry Pi 3, XBee S1 Pro 802.15.4, ciudades inteligentes, cultivos inteligentes, monitorización, sensores, inalámbrico.

Keywords

Arduino UNO, Raspberry Pi 3, XBee S1 Pro 802.15.4, smart-cities, smart-crops, monitoring, sensors, wireless.

Índice general

Índice de tablas y figuras

Capítulo 2:

Capítulo 3:

Capítulo 4:

Capítulo 5:

Capítulo 5 (cont.):

Capítulo 6:

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Mestral Telecomunicaciones SL inició su actividad en 2004 como una empresa de servicios informáticos orientada a las redes y las telecomunicaciones. Su actividad se centra en la gestión y mantenimiento de centrales telefónicas y de comunicaciones, y en el diseño e instalación de redes, principalmente en oficinas. Así mismo, también gestionan servidores a través de los cuales ofrece servicios de hosting a empresas. Actualmente desarrolla su actividad entre el edificio Espaitec I de la Universitat Jaume I y Cumulus, en Villa-real, un incipiente espacio de coworking que alberga a empresas de diversa índole y en donde se ha llevado a cabo la estancia en prácticas.

El proyecto propuesto desde Mestral Telecomunicaciones SL consiste en desarrollar la infraestructura de un sistema de monitorización del entorno mediante un conjunto de nodos equipados con sensores capaces de conectarse entre sí, formando una red, y de transmitir la información recogida a través de estos sensores a un coordinador, el cual, a su vez, transmite dicha información a un servidor con objeto de que pueda ser consultada y analizada a través de la web.

Este proyecto forma parte de una primera fase de inmersión de la empresa en el campo de las *smart-cities* y otros medios susceptibles de ser monitorizados. Se pretende acceder a un nuevo tipo de mercado en auge basado en hacer "inteligentes" a ciertos elementos de nuestro entorno de manera que sean más eficientes y/o ayuden a mejorar la calidad de vida de las personas o la productividad y buen uso de los recursos en empresas e instituciones.

1.2. Objetivos del proyecto

El desarrollo del proyecto cuenta con tres partes diferenciadas: los nodos, el coordinador y la web junto a la base de datos en el servidor. Cada uno de estos elementos que componen la red de monitorización se ha concebido como un bloque general lo más autónomo posible en su desarrollo. Por ello el proyecto puede desglosarse en tres objetivos principales:

- En primer lugar, el desarrollo del firmware que permitirá establecer la comunicación entre los nodos de monitorización Arduino y el coordinador de la red (Raspberry Pi) y enrutar mensajes entre nodos, así como definir la recogida de información a través de los sensores y cómo será transmitida (los frames o mensajes). Esto incluye el desarrollo y/o búsqueda de las bibliotecas necesarias.

- En segundo lugar, la transmisión de los datos adquiridos por el nodo a una base de datos alojada en el servidor de la empresa, que empleará *MongoDB* con el objetivo de crear un gran repositorio de datos acorde al concepto de big data, que permita en un futuro una gestión inteligente de los datos recabados. Se debe desarrollar un software que se ejecute en el coordinador de la red Raspberry Pi y permita que se comunique, además de con los nodos, con la base de datos, desempeñando así la función de puerta de enlace con el servidor.

- Por último, el diseño de una plataforma web como *frontend* que proporcione acceso a la información almacenada en la base de datos de forma textual o gráfica, así como la posibilidad de gestionar el sistema y crear alertas.

Con estos tres bloques trabajando en conjunto, se debe obtener un sistema que permita desplegar nodos independientes en zonas urbanas y/o agrícolas, los cuales, de manera autónoma, deben ser capaces de unirse a la red y transmitir la información recogida por sus sensores, a la vez que son validados contra el servidor para corroborar que se tratan de nodos válidos. Además, los nodos que no puedan alcanzar al coordinador de la red deben poder hacer llegar los datos recogidos a su destino a través de otros nodos vecinos que actúen como enrutadores. Los datos recogidos por el coordinador de la red deben, además, llegar a través de la red hasta el servidor de la empresa de modo que puedan ser integrados en una base de datos que también puede ser consultada a través de una página web.

1.3. Estructura de la memoria

La documentación del desarrollo del proyecto comienza en el *capítulo 2* con una descripción de las tecnologías empleadas y de cómo se unen para dar forma a los componentes del sistema. A continuación, en el *capítulo 3*, se habla de la metodología empleada y la planificación seguida para la consecución del proyecto y se describen las tareas llevadas a cabo. También se presenta un desglose de los recursos necesarios para el desarrollo, así como los costes derivados de los mismos. En el *capítulo 4*, se describen los requisitos técnicos definidos para el proyecto, tanto aquellos referentes a las características técnicas de los componentes del sistema, como del software y de datos. También se muestra el diagrama de casos de uso. El *capítulo 5: Diseño de la arquitectura del sistema* ha sido dividido en tres partes. Se ha realizado una descripción técnica del diseño de la red, del diseño lógico (es decir, del software del sistema) y una descripción de la interfaz. En el *capítulo 6* se han especificado los detalles de la implementación de las funcionalidades más importantes del sistema y un desglose de las pruebas realizadas para verificar el correcto funcionamiento del sistema. Y, por último, en el *capítulo 7* se ha realizado una valoración del cumplimiento de los objetivos, se han propuesto mejoras y se ha incluido una opinión personal acerca del desarrollo de un proyecto de estas características.

Capítulo 2

Descripción del proyecto

En este capítulo describe el punto de partida del proyecto así como las tecnologías hardware y software empleadas para su consecución.

2.1. Definición del sistema

A continuación, se pasa a describir con más detalle la red de monitorización del entorno, el desarrollo de la cual parte desde cero en la empresa. No existen precedentes en la misma en el desarrollo de un proyecto similar, ya que se trata de un prototipo que tiene como objetivo explorar nuevos mercados. Como prototipo, busca tener como funcionalidad fundamental la capacidad de recopilar datos del entorno y permitir su visualización remotamente a través de la web pudiendo variar las tecnologías empleadas desde la versión actual a la que finalmente, llegado el caso, se convertirá en un producto comercial.

Para implementar la red de monitorización descrita, se han empleado las siguientes tecnologías hardware:

- Arduino UNO R3
- Raspberry Pi 3 B con el sistema operativo Raspbian Jessie
- Shield Raspberry Pi - XBee de Cooking Hacks
- Shield Arduino - Xbee
- Xbee Serie 1 Pro 802.15.4
- Adaptador USB para XBee + XCTU
- Antenas de alta ganancia para XBee
- Sensor de temperatura y humedad DTH11 para la toma experimental de datos

Y software:

- C++ como lenguaje de programación para controladores y bibliotecas
- Base de datos MongoDB
- Servidores web Apache y MongoDB server
- Lenguajes web para la construcción del *frontend*
- Software de gestión de alertas
- IDE (Entorno de Desarrollo Integrado) oficial de Arduino y Sublime Text 3 como herramientas de edición para la escritura del código fuente de los programas.

Dichas tecnologías dan lugar a los tres componentes fundamentales del sistema:

1. **Nodo / Nodo arduino**: compuesto por un Arduino, un Shield Arduino - XBee, un XBee con una antena de alta ganancia y uno o varios sensores.

2. **Coordinador / Coordinador de la red / Coordinador Raspberry Pi**: compuesto por una Raspberry Pi, un Shield Raspberry - XBee y un XBee con una antena de alta ganancia.

3. **Frontend / Frontend web / Web**: combina las tecnologías web y el uso de una base de datos MongoDB en el servidor para conformar una plataforma web de gestión del sistema y consulta de datos.

2.2. Descripción de las tecnologías hardware

Arduino UNO R3

Arduino (ver Figura 2.1) es una plataforma de hardware libre que cuenta con su propio IDE de desarrollo con el que es posible implementar programas que permiten la interacción con elementos externos a través de sus pines de E/S usando C++. Además, es posible utilizar adaptadores denominados "*shields*" que permiten incorporar periféricos complejos como, por ejemplo, módulos de comunicación inalámbrica. Estas placas presentan un consumo de energía muy bajo, según mediciones del orden de 46 mA por sí solas, que puede disminuirse utilizando los diferentes modos de sueño o *sleep modes*. Arduino no destaca por su potencia sino por su versatilidad, tal y como se puede ver en el resumen de las características técnicas de la *Tabla 2.1*.

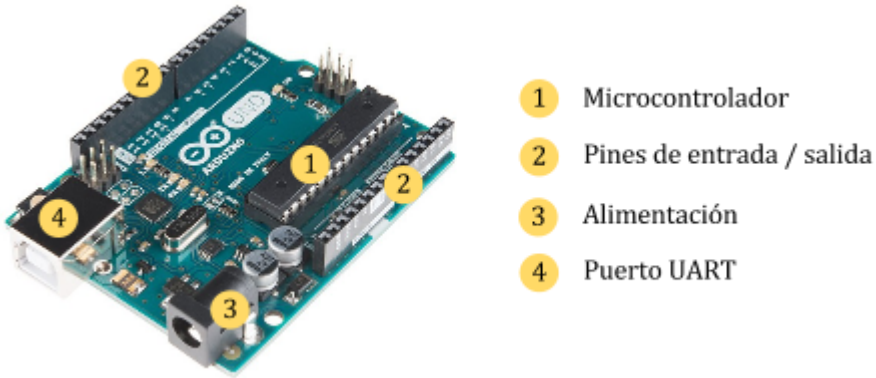


Figura 2.1: Arduino UNO R3

Microcontrolador	Atmega328
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7 - 12V
Voltaje de entrada (Límite)	6 - 20V
Pines para entrada-salida digital	x15 (6 pueden usarse como salida de PWM)
Pines de entrada analógica	x6
Corriente contínua por pin IO	40 mA
Corriente contínua por el pin 3,3V	50 mA
Memoria Flash	32 KB (0,5 KB ocupados por el bootloader(¡)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz
Temperatura de trabajo	-40ºC a 85ºC

Tabla 2.1: Ficha técnica de Arduino Uno R3 [1]

Raspberry Pi 3 B

Raspberry Pi (ver *Figura 2.2*) es un ordenador de placa reducida capaz de ejecutar un sistema operativo completo como Linux o Windows (en realidad versiones adaptadas para consumir pocos recursos). Posee de forma nativa capacidad de conexión WiFi, sin embargo, ha sido necesario emplear un shield para conectar un módulo XBee de manera que pueda comunicarse con los nodos de la red. La conexión WiFi, o bien Ethernet si la ubicación lo permite, ha sido aprovechada para transmitir los datos a través de la red hacia el servidor de la empresa, en donde son almacenados. Las características de la Raspberry Pi 3 B son equiparables a las de un terminal móvil de gama media, suficiente para actuar como coordinadora de la red y puerta de enlace y soportar la carga de múltiples nodos enviando datos. Respecto al sistema operativo, Raspbian Jessie, se trata de una distribución basada en Linux Debian. Ver las características técnicas en la *Tabla 2.2: Ficha técnica de Raspberry Pi 3 B* [2].

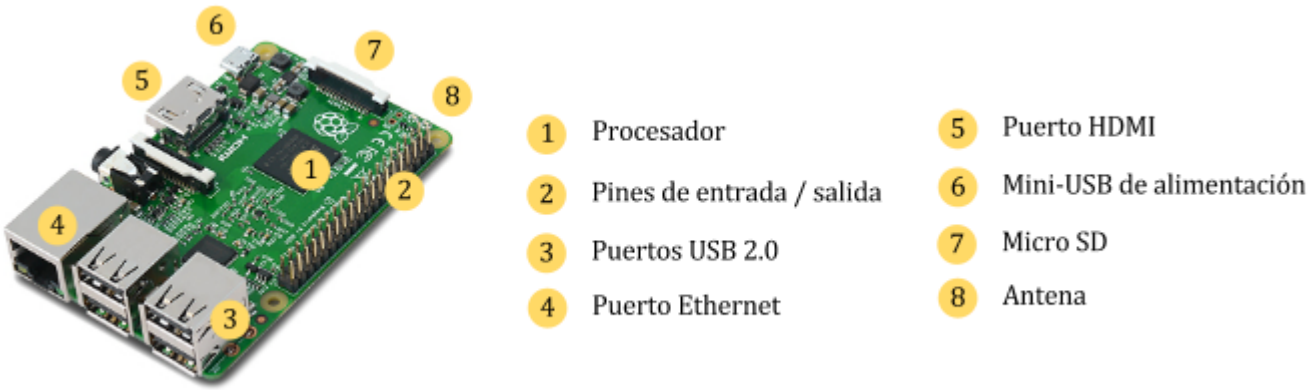


Figura 2.2: Raspberry Pi 3 B

Shield Arduino - XBee

El **shield Arduino - XBee** (ver *Figura 2.3*) tiene como función simplificar la conexión de un módulo XBee S1 Pro a la placa Arduino permitiendo, además, que el resto de pines de E/S sigan disponibles, es decir, que no queden ocultos debajo del mismo. Posee además un interruptor que permite alternar el control del puerto serie entre Arduino y el módulo XBee S1 Pro.

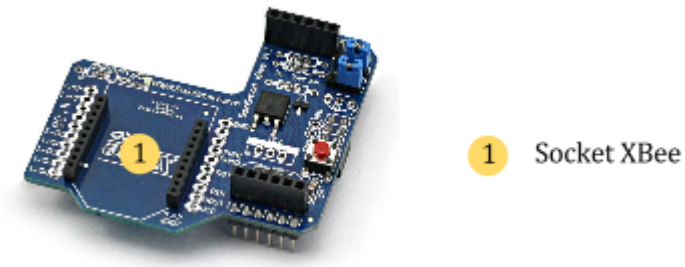


Figura 2.3: Shield Arduino - XBee

Procesador	BCM2837 64 bits x4 nucleos ARMv8 a 1,2 GHz
Chip gráfico	Dual Core VideoCore IV Multimedia
Voltaje de operación	1,5 - 5V
Voltaje de entrada máximo	3,3V
Corriente contínua por pines GPIO	16 - 50 mA
RAM	1 GB
Pines de entrada-salida	x27 pines GPIO + UART + Bus I2C + Bus SPI
Puertos USB 2.0	x4
Puerto Full HDMI	x1
Puerto Ethernet	x1
Audio jack 3.5 mm	x1
WiFi	802.11n
Bluetooth	4.1 y LE (*Low Energy* / Baja energía)
Slot micro SD	x1 hasta 32 GB
Temperatura de trabajo	-40ºC a 85ºC

Tabla 2.2: Ficha técnica de Raspberry Pi 3 B

Shield Raspberry Pi - XBee

El **shield Raspberry Pi - XBee de Cooking Hacks** (ver *Figura 2.4*) ha sido ideado para ser usado junto a la biblioteca arduPi como adaptador de proyectos Arduino a Raspberry Pi. Esto es, posee entradas y salidas que se corresponden con los presentes en las placas Arduino además de un slot específico para XBee. A su vez la biblioteca arduPi proporciona una capa de abstracción para que los códigos implementados para Arduino puedan funcionar en la Raspberry Pi sin modificar el uso de ciertas funcionalidades, como, por ejemplo, el puerto serie o funciones como millis() (que permite obtener una marca de tiempo en milisegundos) o delay(*milisegundos*) que detiene el progreso de la ejecución durante los milisegundos indicados como parámetro. El shield no requiere el uso obligado de esta biblioteca para funcionar.

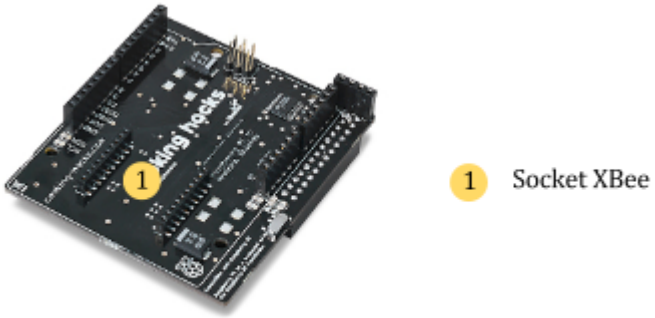


Figura 2.4: Shield Raspberry Pi - XBee

XBee S1 Pro 802.15.4

Xbee Serie 1 Pro 802.15.4 (ver *Figura 2.5*) es un módulo de comunicación inalámbrica que permite realizar conexiones punto a punto o multipunto (*broadcast*) y que está basado en el estándar IEEE 802.15.4 [4]. En la *Tabla 2.3* se muestran las características del mismo. Este estándar define las capas del nivel físico, de control de acceso al medio y de enlace para redes WPAN (redes inalámbricas de área personal) con bajas tasas de transmisión de datos. La versión Pro transmite una señal con una potencia de 100 mW frente a los 10 mW del modelo convencional, lo que le proporciona, en condiciones de visibilidad óptima, un alcance de 1,6 km con una velocidad de transmisión de datos de hasta 250 Kbps. Su consumo cuando se encuentra transmitiendo es de 215 mA y de 55 mA cuando recibe datos. El modelo simple tiene un consumo menor, del orden de 45 a 50 mA, no obstante apenas alcanza los 90 metros de distancia, de modo que con la versión Pro los nodos tienen mayores posibilidades de alcanzar a otros sin necesidad de enrutar a sus paquetes a través de nodos vecinos, evitando así sobrecargar la red y que los nodos enrutadores pasen demasiado tiempo despiertos consumiendo energía. XBee también soporta el cifrado de datos en la capa de acceso al medio para hacer más seguras las comunicaciones [3].

- Tasas de transmisión de datos soportadas de 20 kbps, 40 kbps y 250 kbps
- Modos de direccionamiento con direcciones de 16 y 64 bits
- Soporte para dispositivos en los que la latencia es crítica
- CSMA-CA (acceso múltiple por detección de portadora y prevención de colisiones)
- Protocolo para transferencia de datos fiables totalmente negociado
- Administración de energía para asegurar un bajo consumo energético
- 16 canales en la banda ISM de 2.4 GHz, 10 para 915 MHz (EEUU) y uno para 868 MHz.

Tabla 2.3: Características del estándar IEEE 802.15.4

Por otra parte, como se ha mencionado en el párrafo anterior, entre las características de los módulos XBee S1 802.15.4 [5] sólo se incluyen el envío de mensajes directamente a otros módulos o a varios a través de un broadcast por lo que no tienen la capacidad de enrutar mensajes por sí solos. Por tanto, esta característica ha tenido que ser implementada por software, ya que pese a la gran distancia que son capaces de abarcar se deben tener en cuenta posibles obstáculos que impidan que un nodo tenga acceso al coordinador de la red o que éste se encuentre muy lejos.

En la *Tabla 2.4* se pueden ver más detalles sobre las características técnicas del módulo XBee.

Figura 2.5: XBee S1 Pro 802.15.4

Frecuencia	2.4 GHz
Canales	16
Máxima línea de visión	1,6 Km
Tasa de transmisión de datos	250 kbps
Protocolo de comunicación	802.15.4
Máxima transmisión de potencia	100 mW
Máxima sensibilidad del receptor	-100 dBm
Consumo transmitiendo	215 mA
Consumo recibiendo	55 mA
Consumo en reposo	< 10 uA
Temperatura de funcionamiento	-40ºC a 85ºC

Tabla 2.4: Ficha técnica de XBee S1 Pro 802.15.4

Adaptador USB para XBee + XCTU

El **adaptador USB para XBEE** (ver *Figura 2.6*) permite conectar los módulos XBee a un ordenador a través del puerto USB y, mediante la aplicación **XCTU**, establecer la configuración por defecto de los módulos (ID de la red, dirección destino si procede, nombre, tipo de comunicación, consumo y alcance...), hacer pruebas o actualizar el firmware entre otras cosas. También puede ser empleado como adaptador para conectar el módulo XBee manualmente, mediante cableado, a los pines de entrada y salida (RX y TX) correspondientes al puerto serie de la placa Arduino o Raspberry a la que se desee conectar.

Figura 2.6: Adaptador USB para XBee

Antena de alta ganancia para XBee

Las **antenas de alta ganancia** (ver *Figura 2.7*) permiten explotar el máximo potencial de transmisión de los módulos XBee y alcanzar las distancias descritas en el apartado referente al XBee S1 Pro 802.15.4, que describe las características de dichos módulos.

Figura 2.7: Antena de alta ganancia

Sensor DHT11

El **sensor de temperatura y humedad DHT11** (ver *Figura 2.8*) permite medir la humedad y temperatura ambientales. Se trata de un sensor muy básico ya que sólo es capaz de leer temperaturas positivas y en rangos de enteros con una precisión de ±2ºC así como rangos de humedad que van desde el 20% al 80% de humedad ambiental relativa. Él propósito de estos sensores es constatar la capacidad de los nodos para recolectar datos, ya que se pretende que el producto final disponga de varias combinaciones de sensores mucho más avanzados que se adapten a las necesidades del cliente y/o entorno en el que sean desplegados. En la *Tabla 2.5* se detallan las características técnicas de este sensor.

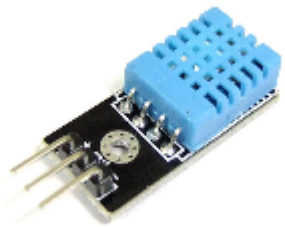


Figura 2.8: Sensor DHT11 de temperatura y humedad

Por ejemplo, un nodo pensado para monitorizar un cultivo podría disponer de sensores independientes de humedad, temperatura, presión atmosférica y detección de lluvia.

Voltaje de operación	3,3 y 5 V
Consumo	2,5 mA durante la lectura de datos
Rango de temperatura medible	0ºC a 50ºC
Precisión en la medición de temperaturas	5% (±2ºC)
Rango de humedad ambiental medible	De 0ºC a 25ºC del 30 al 90% De 25ºC a 50ºC del 20 al 90% A los 50ºC del 20 al 80%
Precisión en la medición de humedad	5%
Tasa mínima de muestreo	1 Hz (una vez cada segundo)
Pines	4 (Vcc, tierra, datos y uno sin uso)

Tabla 2.5: Ficha técnica del DHT11 [6]

2.3. Descripción de las tecnologías software

C++: Software de control y bibliotecas

El lenguaje de programación utilizado para la implementación del software de los nodos Arduino y Raspberry Pi ha sido C++. La principal razón para utilizar este lenguaje de programación es poder implementar una biblioteca única para gestionar las comunicaciones a través de los módulos XBee que pueda ser compartida por ambas plataformas, ya que los programas para Arduino se escriben únicamente en este lenguaje de programación mientras que Raspberry Pi puede trabajar con un gran número de lenguajes de programación siempre que disponga del compilador oportuno.

El software que controla a los nodos y al coordinador de la red emplea las funciones implementadas en las bibliotecas desarrolladas para describir el comportamiento del dispositivo.

En el caso de los nodos Arduino el software desarrollado está compuesto por un programa principal con las funciones de detección de vecinos, búsqueda del coordinador de la red y procesamiento de frames y un módulo con las funciones de lectura de datos de los sensores instalados. Es decir que este módulo (denominado "SenseX" donde "X" indica de que set de sensores se trata) varía en función de los sensores incorporados al nodo Arduino abstrayendo al programa principal de la lógica o particularidades de la lectura de datos de los mismos.

El software desarrollado para el coordinador de la red, además, lleva a cabo consultas a la base de datos para validar a los nodos que solicitan unirse en la red o enviar datos, gestiona un log de eventos y almacena los datos recibidos desde los nodos en la base de datos del servidor.

En resumen, las bibliotecas que se han escrito y/o utilizado en el desarrollo del proyecto son las siguientes:

- **XBee**: biblioteca implementada específicamente para el proyecto y utilizada para gestionar el envío y recepción de mensajes mediante el módulo XBee empleando el modo API2 (frame estructurado con escape de caracteres) [3].
- **SerialManager**: biblioteca implementada específicamente para el proyecto cuyo objetivo es abstraer el funcionamiento del puerto serie para cada una de las dos plataformas, Arduino y Raspberry Pi. Es utilizada por la biblioteca XBee para leer y escribir en el puerto serie, que es el utilizado por los módulos XBee para las comunicaciones.
- **SnoozeLib**: biblioteca utilizada en Arduino para llevar al microcontrolador a un modo de bajo consumo y de este modo ahorrar batería. Ha sido obtenida de los repositorios públicos de Arduino.
- **QueueList**: biblioteca utilizada en Arduino para gestionar colas FIFO. Es utilizada para encolar los mensajes recibidos desde otros nodos de la red cuando responden a una petición de un nodo buscando a un nodo vecino que se convierta en su enrutador. Ha sido obtenida de los repositorios públicos de Arduino.
- **MongoDB C++ / PHP driver**: proporcionados por MongoDB, estos drivers permiten conectar con la base de datos y realizar transacciones sobre ella desde C++ y PHP.

Base de datos: MongoDB

MongoDB [7] es una base de datos noSQL[1] que almacena las estructuras de datos en documentos con un esquema similar al formato JSON al que denomina Binary JSON o BSON [7]. Este formato permite que la integración de los datos sea más sencilla y rápida ya que, entre otras cosas, permite estructuras y documentos embebidos dentro de otros, utiliza los tipos de datos definidos en C, por lo que es fácilmente accesible para muchos lenguajes de programación, es escalable y ligero, soporta balanceo de carga cuando se encuentra distribuido entre varios servidores y soporta búsquedas por rangos y el uso de expresiones regulares, así como consultas del tipo "GROUP BY" de SQL no soportadas por otras bases de datos noSQL.

No obstante, tiene algunas desventajas respecto a los sistemas de bases de datos tradicionales y otras alternativas noSQL del mercado, ya que no implementa las propiedades ACID (*Atomicity, Consistency, Isolation and Durability* o Atomicidad, Consistencia, Aislamiento y Durabilidad). Esto se traduce en problemas de consistencia, ya que las lecturas estrictamente consistentes ven versiones obsoletas de los documentos[2], porque la única forma de realizar escrituras verdaderamente concurrentes es entre documentos distintos puesto que los bloqueos de escritura se realizan a nivel de documento. Además, las escrituras no garantizan su durabilidad o veracidad, puesto que MongoDB devuelve la confirmación de la escritura antes de que se haya almacenado de forma permanente en todos los documentos y réplicas.

Servidores web: Apache y MongoDB server

Apache es un servidor web HTTP de código abierto compatible con sistemas operativos basados en UNIX y Windows. Se considera seguro y eficiente y muy versátil por la gran cantidad de extensiones que soporta. Su configuración y puesta en marcha resulta muy sencilla por lo que es una excelente opción para albergar una página web como el frontend de administración de la red de este proyecto.

Por otra parte, MongoDB server (denominado *mongod*), una utilidad que se instala junto a MongoDB, se ejecuta en el servidor de la empresa para atender a las peticiones de conexión y acceso a los datos de la base de datos por parte del coordinador (remotas) y de la web (locales).

---

[1] "*NoSQL [...] engloba a una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes, el más destacado es que no usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad), y habitualmente escalan bien horizontalmente. Los sistemas NoSQL se denominan a veces "no sólo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL.*" Fuente: https://es.wikipedia.org/wiki/NoSQL (Wikipedia, 2016)

[2] El equivalente a un "Documento" en una modelo de bases de datos relacional SQL es "Fila".

Frontend web: HTML, CSS, Javascript y PHP

Estas tecnologías web se han utilizado para dar forma al frontend. Puesto que el proyecto no está centrado en el desarrollo de la plataforma web sino en la infraestructura que sustenta la obtención de los datos, no se ha profundizado en la posibilidad de incorporar y/o utilizar soluciones más avanzadas.

El uso de JavaScript incluye la API de Google Maps para generar y mostrar los mapas utilizados en el *frontend* web para ubicar a nodos y coordinadores. Dicha característica se describe en detalle en el apartado 5.3: *Diseño de la interfaz*, del *capítulo 5*.

Software de gestión de alertas

Este programa desarrollado en C++ se aloja en el servidor y se encarga de monitorizar aquellos datos de la base de datos implicados en una alerta programada consultándolos cada cierto periodo de tiempo, y de realizar el aviso pertinente. En la versión actual del proyecto un aviso implica el envío de un correo electrónico.

Herramientas de desarrollo: IDE de Arduino y Sublime Text 3

Para el desarrollo del software que controla a los nodos se ha utilizado el IDE de Arduino distribuido de forma gratuita a través de la página web oficial[3]. Este IDE permite escribir los programas, depurarlos, cargarlos en las placas Arduino y visualizar posibles entradas y salidas de datos mediante una consola que muestra todo lo que se envía y recibe a través del puerto serie.

Para el desarrollo del software del coordinador de la red Raspberry Pi se ha empleado Sublime Text 3[4]. Este programa es un sencillo editor de texto libre que resalta las palabras clave del lenguaje de programación que esté siendo utilizado además de sugerir funciones y variables utilizadas en otras partes del código.

[3] Arduino - Software (IDE): https://www.arduino.cc/en/Main/Software

[4] SublimeText 3: https://www.sublimetext.com

Capítulo 3

Planificación del proyecto

En este capítulo se va a describir la metodología usada para llevar a término el proyecto, así como las tareas que se han llevado a cabo para su realización y cómo han sido planificadas. También se incluye una justificación de las desviaciones que se han producido respecto a la planificación inicial y un apartado en el que detalla el coste de los recursos empleados en el proyecto.

3.1. Metodología

Para diseñar el sistema de monitorización del entorno desarrollado en este proyecto e integrar cada uno de los elementos que lo componen se ha utilizado la metodología de diseño denominada **Top-down** [8], tal y como se puede ver en la *Figura 3.1*. Esta metodología se caracteriza por definir la funcionalidad general del sistema e ir concretando sus componentes, funciones y relaciones a lo largo de sucesivas etapas, de forma que en cada una de ellas se obtiene una versión más concreta del sistema hasta llegar a una versión funcional y completa del mismo. Para ello, se descompone el proceso de desarrollo en cinco etapas consecutivas, las cuales son:

1. Toma de requisitos, que pueden ser funcionales o no funcionales: se utilizan para confeccionar unas especificaciones precisas y realistas que permitan diseñar la arquitectura del sistema. Se suele realizar mediante la cumplimentación de un formulario que recoge cuestiones como: coste del producto, dimensiones físicas y peso, consumo energético, propósito del sistema, funciones, entradas/salidas o prestaciones.

2. Descripción de las especificaciones: se trata de una concreción técnica y sin ambigüedades de los requisitos. Cualquier requisito considerado inviable (no funcional) queda descartado en esta etapa.

3. Diseño de la arquitectura: se especifica cómo hacer lo indicado en la descripción de las especificaciones. Consiste en un esquema global del sistema sobre el que se diseñarán o seleccionarán los componentes. Incluye tanto el esquema general del hardware como el del software.

4. Desarrollo técnico del proyecto: en esta etapa se materializan y/o adquieren los componentes hardware y software del sistema. Tanto si se trata de componentes diseñados específicamente como si han sido adquiridos, debe realizarse la unión entre estos y las modificaciones oportunas en la programación para que se relacionen entre sí de acuerdo al diseño de la arquitectura.

5. Integración y verificación de los componentes: se realiza el montaje de los componentes del sistema y se llevan a cabo pruebas para depurar posibles errores surgidos del desarrollo por separado de los componentes.

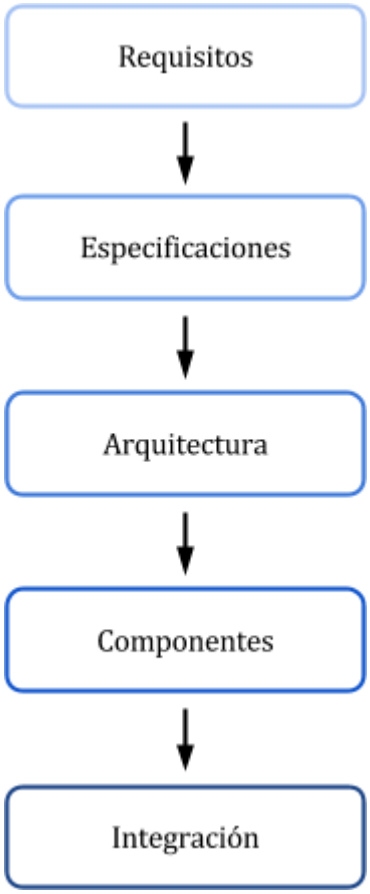


Figura 3.1: Esquema de la metodología Top-down

En cada una de estas etapas es prioritario tener en cuenta las prestaciones del sistema. Se debe prestar especial atención al rendimiento y restricciones temporales, así como al consumo de energía (por ejemplo, en el caso que nos ocupa, especificando los períodos de envío y recepción de mensajes y el tiempo que los nodos permanecen dormidos[5]). También se debe tener en consideración el coste de fabricación del producto en cada una de las etapas. Por último, resulta imprescindible comprobar que en cada una de ellas se cumple con las especificaciones definidas y los objetivos generales del sistema para evitar que el resultado final se desvíe de lo esperado.

La estrategia Top-down es especialmente eficaz cuando se tienen conocimientos en el diseño de sistemas similares o bien se tienen los conocimientos e información necesarios para llevar a cabo la implementación del sistema.

[5] Es decir, en modo de bajo consumo.

3.1.1 Puesta en práctica de la metodología

La metodología Top-down abarca el diseño de un sistema desde el planteamiento inicial hasta la materialización del mismo apoyándose en las etapas previas del proceso para obtener resultados en cada nueva etapa. No obstante, en el proyecto que nos ocupa los componentes hardware del sistema han sido establecidos por la empresa al inicio del proyecto, por lo que han actuado como condicionante en la descripción de la especificación técnica del proyecto. Se ha tenido que contrastar si las características técnicas de cada uno de ellos se adecuaban a los requisitos indicados por la empresa para establecer cuáles de ellos pasaban a formar parte de la especificación y cuáles no. Del mismo modo, han condicionado el diseño de la arquitectura, que ha tenido que adaptarse a dichos componentes, y el diseño y selección del resto de componentes hardware y software, que han tenido que ser seleccionados de modo que fueran compatibles.

A pesar de esto, el uso de esta estrategia ha resultado adecuado ya que el hecho de no tener que definir con exactitud la forma final de cada uno de los componentes del proyecto desde el inicio permite una adaptación más dinámica ante eventos inesperados o problemas técnicos no previstos en la concepción inicial del proyecto o derivados de la falta de experiencia en el desarrollo de un proyecto de estas características.

3.2. Planificación del desarrollo

3.2.1 Definición de tareas

Para completar el proyecto se han llevado a cabo las siguientes tareas, las cuales se presentan agrupadas en etapas equivalentes a las de la metodología Top-down:

- Planteamiento del proyecto y toma de requisitos

    1. Definir el proyecto con el tutor y el supervisor: el supervisor enuncia la idea general del proyecto y se procede a la toma de requisitos.

    2. Definir la metodología de trabajo y la documentación a utilizar: se ha procedido a revisar y buscar en la red toda la documentación que deberá ser utilizada. Así mismo, se ha estudiado la mejor forma de afrontar el desarrollo de un proyecto de estas características para un estudiante en prácticas.

    3. Definir las tareas y estimar el tiempo de ejecución: se han definido de forma general las tareas necesarias para completar el proyecto y se les ha asignado un tiempo de ejecución.

- Preparación previa

    4. Documentarse sobre los elementos que conforman la red de sensores: se ha procedido a revisar y estudiar toda la documentación facilitada por la empresa y, de forma complementaria, en la web para comprender el funcionamiento de las tecnologías utilizadas para implementar el proyecto.

    5. Instalar el software de trabajo: se ha instalado el software necesario para desarrollar el código del proyecto. Incluye el gestor de base de datos y los IDE además de actualizaciones y otro software para Raspberry Pi.

    6. Familiarizarse con las herramientas de trabajo software y hardware: se ha reservado un periodo de tiempo para familiarizarse con las tecnologías más novedosas y poder realizar pruebas con ellas con el fin de comprender su funcionamiento y cómo tratar con ellas. Se han creado pequeños programas en C++ para comprobar cómo funcionan las entradas y salidas de los nodos Arduino, para transmitir mensajes en texto plano a través de los módulos XBee y se han realizado consultas de prueba e inserciones sobre la base de datos en MongoDB.

- Definición de las especificaciones del proyecto y diseño de la arquitectura

    7. Definir las especificaciones técnicas del proyecto: con los conocimientos adquiridos a través de la documentación y las pruebas se han determinado las características del sistema y se han rechazado aquellos requisitos inviables. Por ejemplo, dadas las características del material disponible se ha tenido que descartar la posibilidad de realizar actualizaciones vía OTA (Over The Air) del firmware ya que no se disponía de los módulos necesarios para que los nodos Arduino pudieran albergar una tarjeta microSD en la que almacenar las actualizaciones.

    8. Diseñar la arquitectura del proyecto: se ha dado forma a los nodos y al coordinador de la red con los distintos elementos y se ha procedido a estudiar qué elementos software eran necesarios para cumplir con el propósito del proyecto. Se ha determinado que cada elemento contará con un programa principal dedicado a gestionar las tareas más específicas y que para el control de los módulos XBee y de los modos sueño de los nodos Arduino se utilizarán bibliotecas externas.

- Desarrollo técnico del proyecto. Se han desglosado las tareas de forma que al finalizar cada una de ellas se disponga de una parte funcional del sistema lista para ser integrada al conjunto de partes que lo forman.

    9. Diseñar y crear la base de datos en MongoDB: se ha diseñado una base de datos con colecciones[6] para albergar usuarios, datos sobre sensores, información sobre nodos y configuraciones de alertas y rutas. A continuación se ha procedido a crear las colecciones.

[6] El equivalente a “Colección” en un modelo de bases de datos relacional SQL es “Tabla”

10. Establecer la configuración inicial del servidor: se ha comprobado que los permisos para MongoDB y de la configuración de red y de Apache son correctas y permiten conectividad con una aplicación del exterior.

11. Desarrollo de la biblioteca de comunicaciones: se ha desarrollado una biblioteca en C++ para el envío y recepción de mensajes con los módulos inalámbricos XBee y otra, SerialManager, para abstraer la gestión del puerto serie en Arduino y Raspberry Pi.

12. Desarrollar el software de los nodos Arduino: se ha implementado el software controlador de los nodos Arduino que describe el comportamiento del nodo, la capacidad de enrutar y funciones específicas para comunicarse con el coordinador.

13. Desarrollar el software de la puerta de enlace Raspberry Pi: se ha desarrollado la versión de la biblioteca SerialManager para gestionar el puerto serie en Raspberry Pi. También se ha desarrollado el software controlador de Raspberry Pi para hacer las funciones de coordinador de la red y puerta de enlace con el servidor. Esto incluye el procesamiento de los mensajes que llegan desde los nodos y la comunicación remota con la base de datos alojada en el servidor.

14. Implementar el frontend: se ha diseñado e implementado una página web para consultar los datos de la base de datos del servidor y administrar el frontend.

15. Implementar software de gestión de alertas en el servidor: se ha desarrollado la aplicación encargada de gestionar los avisos relacionados con las alertas programadas.

- Integración y verificación de los componentes

16. Realizar tests de comunicaciones y captura de datos: se han realizado diversas pruebas para corroborar que los nodos eran capaces de comunicarse entre sí para el enrutado entre nodos y con el coordinador para poder ser incluidos en la red y para realizar la transmisión de datos. También se ha comprobado si los datos recogidos por los nodos llegaban al coordinador y a la base de datos del servidor.

17. Comprobar la visualización de los datos y gestión de alertas del frontend: se ha comprobado que los datos almacenados en la base de datos son accesibles desde la web y se mostraban correctamente. También se ha comprobado que pueden realizarse modificaciones en la base de datos a través de las secciones de administración del *frontend.*

18. Redactar y entregar el informe final para la empresa: se ha redactado un informe para el supervisor de la empresa resumiendo el trabajo realizado así como una descripción de los componentes del sistema, de su funcionamiento y del funcionamiento del sistema en general con sus capacidades, funcionalidades y limitaciones.

3.2.2 Planificación temporal

Las tareas descritas en el punto anterior se han organizado de forma cronológica tal y como aparece en la *Tabla 3.1*. Además, en esta tabla han sido incluidas las tareas a realizar para la universidad, tales como la redacción de informes quincenales o la presentación del proyecto. Con el asesoramiento del supervisor de la empresa, se le ha asignado a cada una de las tareas el número de horas considerado más adecuado para llevarla a término. Además, se ha prestado especial atención en cubrir las 300 horas requeridas por la universidad a razón de una dedicación en la empresa de 25 horas semanales ó 5 horas diarias. En la *Figura 3.2* se puede observar un diagrama de Gantt en el que se representan de forma gráfica las tareas distribuidas a lo largo del periodo de tiempo correspondiente a la estancia en prácticas.

Nº	Descripción	Tiempo (h)	Dependencias
1	**Planteamiento del proyecto**	**11 h**	
1.1	Definir el proyecto con el tutor y el supervisor	1	
1.2	Definir la metodología de trabajo y la documentación a utilizar	5	1.1
1.3	Definir las tareas y estimar el tiempo de ejecución	5	1.2
2	**Preparación previa**	**35 h**	
2.1	Documentarse sobre las tecnologías que conforman la red de sensores: Arduino, Raspberry Pi, MongoDB y lenguajes web	25	1.1
2.3	Instalar el software de trabajo	5	1.1
2.3	Familiarizarse con las herramientas de trabajo software y hardware	5	2.3
3	**Definición del proyecto**	**15 h**	
3.1	Definir las especificaciones técnicas del proyecto	5	2.2
3.2	Diseñar la arquitectura del proyecto	10	3.1
4	**Desarrollo técnico del proyecto**	**200 h**	
4.1	**Puesta a punto de la infraestructura de apoyo**	**30 h**	
4.1.1	Diseñar y crear la base de datos en MongoDB	15	3.2
4.1.2	Establecer la configuración inicial del servidor	15	3.2
4.2	**Desarrollo de la biblioteca de comunicaciones**	**80 h**	
4.2.1	Desarrollar la biblioteca XBee y SerialManager	80	3.2
4.3	**Desarrollo del software de los sensores**	**25 h**	
4.3.1	Desarrollar el software de los nodos Arduino	25	4.2.1
4.4	**Desarrollo del software de gestión de la puerta de enlace**	**25 h**	
4.4.1	Desarrollar el software de la puerta de enlace Raspberry Pi	25	4.2.1

4.5	**Desarrollo del frontend**	**40 h**	
4.5.1	Implementar el frontend	20	4.1.2
4.5.2	Implementar software de gestión de alertas en el servidor	20	4.1.2
5	**Integración y verificación de los componentes**	**15 h**	
5.1	Realizar tests de comunicaciones y captura de datos	10	4.3.1
5.2	Comprobar la visualización de los datos y gestión de alertas del frontend	5	4.4.2
6	**Informe final para la empresa**	**24 h**	
6.1	Redactar informe final para la empresa	24	5.2
6.2	Entregar informe final a la empresa	0	6.1
7	**Documentación y presentación del TFG**	**158,15 h**	
7.1	Redactar y documentar la propuesta técnica	10	3.2
7.2	Entregar la propuesta técnica	0	7.2
7.3	Redactar informes quincenales	8	7.2
7.4	Redactar la memoria	100	1.1
7.5	Entregar la memoria	0	7.4
7.6	Confeccionar y preparar la presentación	40	7.5
7.7	Realizar la presentación oral	0,15	7.6

Tabla 3.1: Desglose de tareas y asignación del tiempo de ejecución en horas

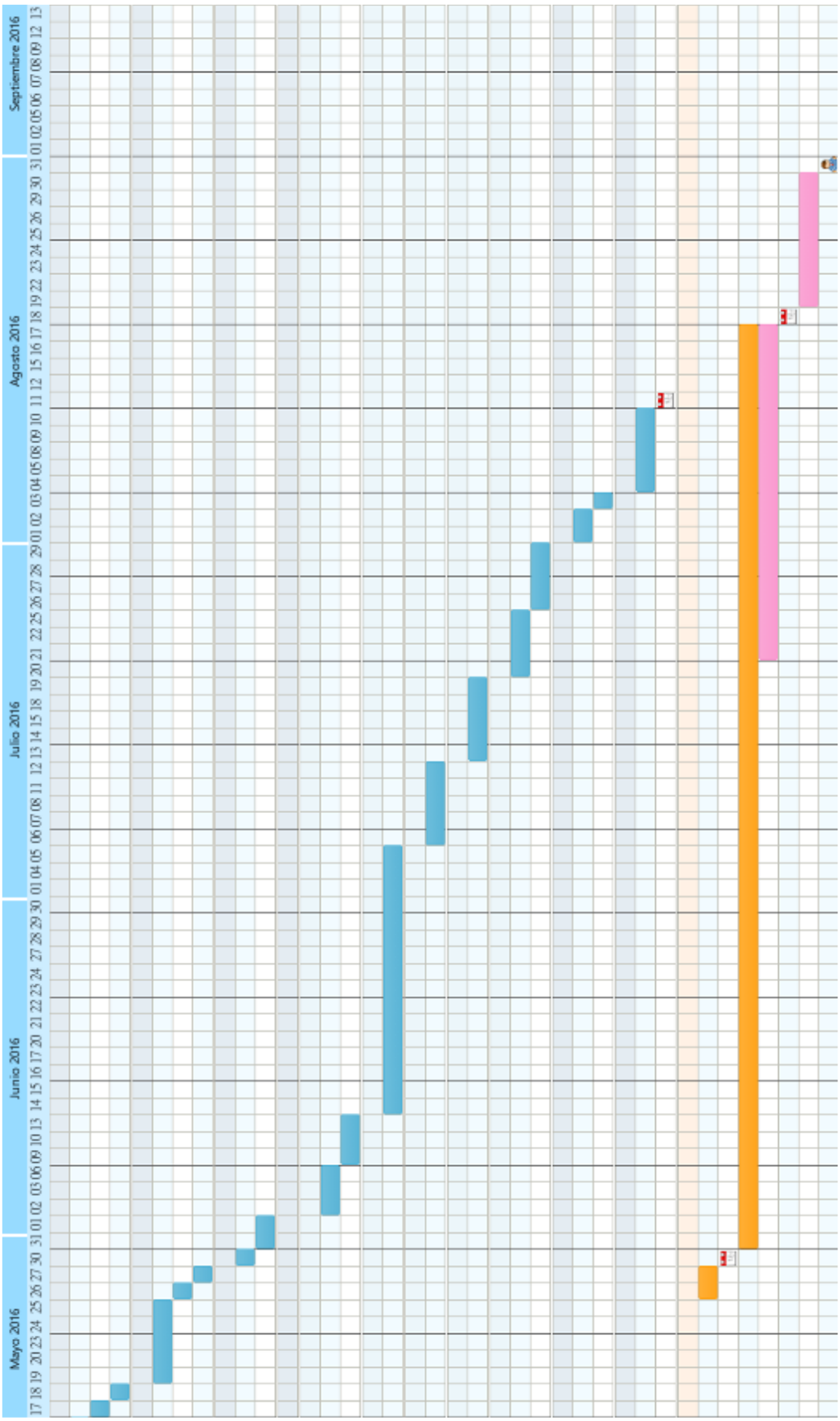


Figura 3.2: Diagrama de Gantt de la planificación del proyecto

3.3. Estimación de recursos y costes del proyecto

Los recursos necesarios para desarrollar el proyecto incluyen las herramientas de trabajo (H) y los componentes (C) para realizar el montaje y testeo del sistema, los cuales han sido desglosados en la siguiente tabla:

Tipo	Recurso	Unidades	Coste	Total
H	Ordenador de trabajo	1	-	-
H	IDE de desarrollo para Arduino	1	0,00€	0,00€
H	Sublime Text 3	1	0,00€	0,00€
C	Arduino (Genuino) UNO	2	20,00€	40,00€
C	Shield Arduino - XBee	2	15,00€	30,00€
C	Raspberry PI 3 B	1	41,95€	41,95€
C	Shield Raspberry Pi - XBee	1	40,00€	40,00€
C	XBee Serie 1 Pro 802.14.5	3	21,95€	65,85€
C	Antena de alta ganancia XBee	3	12,00€	36,00€
C	Adaptador USB XBee	1	7,83€	7,83€
C	Sensor de humedad y temperatura DHT11	2	2,93€	5,86€
C	Cables pack 120 uds: macho-macho, macho-hembra, hembra-hembra	1	2,39€	2,39€
C	Resistencias de 220 Ohmios pack 100 uds	2	0,62€	0,62€
C	Protoboard	2	0,96€	1,92€
C	Caja estanca para pruebas en el exterior	3	1,95€	5,85€

Tabla 3.2: Desglose de costes del proyecto

El coste de los recursos materiales empleados suma un total de **278,27€**.

Por otra parte, el coste estimado de las horas dedicadas al desarrollo del proyecto por parte de un programador junior durante 3 meses, a 20 días por mes, a razón de 5 horas diarias, es decir, 300 horas a 6€/hora[7] es de **1800€**.

Una vez definidos todos los recursos necesarios es posible estimar el coste del proyecto. Así pues, si al coste de los recursos materiales se le suman las horas invertidas en el desarrollo, obtenemos que el coste total del proyecto asciende a un total de **2078,27€**.

[7] Monto calculado partiendo de la base de que a fecha del 7 de agosto de 2016 la media de un sueldo de programador junior, sin tener en cuenta especializaciones, es de unos 18.432€ al año, haciendo un total de 1000€ al mes o 9,6€/hora para una jornada completa de 8 horas.

3.4. Seguimiento del proyecto

Todas las fases de este proyecto en general han sido supervisadas por el tutor y el supervisor de la empresa mediante informes quincenales en los que se han descrito las tareas llevadas a cabo durante la quincena así como los objetivos a cumplir para el siguiente período.

Respecto a la parte del desarrollo técnico, se han establecido unos objetivos funcionales para cada bloque, así como a qué entradas debe responder y qué salidas debe producir para el siguiente bloque o bloques. Tras la consecución de un bloque (por ejemplo, del software que controla a los nodos Arduino) se han realizado pruebas individuales con el fin de comprobar que dicha parte del sistema es capaz de responder a las entradas definidas y producir la salida esperada.

No obstante, ha habido algunos problemas técnicos que si bien no han afectado a las tareas a llevar a cabo o al orden de ejecución, sí que han conllevado leves retrasos en el tiempo total que se había previsto para algunas de las tareas.

El principal inconveniente surgió tras diseñar y programar el software que controla al coordinador de la red y realizar las comprobaciones oportunas. Sucedió que los resultados no eran los esperados, el coordinador de red Raspberry Pi tan solo recibía algunos de los mensajes y los que envíaba no llegaban a los nodos Arduino.

Para solucionar este imprevisto, en primer lugar, se comprobó si el módulo XBee se encontraba en buen estado. Para ello, se conectó al adaptador USB para XBee y se comprobó el firmware y la configuración utilizando el programa XCTU. A continuación, se llevaron a cabo algunas pruebas utilizando las herramientas de envío de mensajes de este software. Tras confirmar que el módulo XBee no era defectuoso, se procedió a conectarlo, colocado en el adaptador, mediante cables a los puertos correspondientes al puerto serie de la Raspberry Pi. Con esta configuración, las transmisiones se realizaban correctamente si se mantenía la conexión con el ordenador a través del puerto mini-USB del adaptador y no sucedía de igual forma si se alimentaba a través de la Raspberry Pi. Por tanto, se llegó a la conclusión de que la alimentación recibida por Raspberry Pi a través del puerto USB del ordenador utilizado para el desarrollo del proyecto no era suficiente. La solución definitiva consistió en utilizar un cargador de móvil compatible para alimentar al coordinador Raspberry Pi.

Capítulo 4

Análisis del sistema

En este capítulo se realiza un análisis de la red de monitorización del entorno describiendo los requisitos del sistema y los casos de uso, y cómo se combinan entre sí para proporcionar la funcionalidad requerida.

4.1 Introducción

El proyecto llevado a cabo no ha sido motivado por un cliente sino que ha sido la propia empresa, quien ha incentivado su desarrollo desde dentro. Por ello, en este caso los requisitos que se van a describir no proceden de un cliente externo particular o empresa.

Los requisitos recabados se pueden clasificar en tres tipos atendiendo al ámbito que pretenden definir:

- Requisitos técnicos: se trata de requisitos no funcionales que recogen aspectos básicos de la operatividad del sistema.

- Requisitos funcionales: expresan las funciones software y de gestión de los datos del sistema y sus componentes.

- Requisitos de datos: recogen aquellas cuestiones relacionadas con los datos que recopilará y almacenará el sistema.

4.2 Requisitos técnicos

La red de sensores debe poder desplegarse en cualquier entorno (dentro de los límites de los componentes en cuanto a temperatura y humedad máxima soportada), por tanto los nodos y el coordinador de la red deben ser portables, autónomos y autosuficientes. Tampoco debe existir un límite en la cantidad de nodos a gestionar. Además, los nodos y el coordinador de la red deben ser capaces de transmitir información a grandes distancias. En el caso de los nodos exclusivamente de forma inalámbrica, mientras que el coordinador deberá ser capaz de adaptarse a la infraestructura disponible y ser capaz de transmitir datos al servidor de forma alámbrica (ethernet) o inalámbrica (WiFi). Los tiempos de estas comunicaciones deben estar dentro de un margen aceptable que permita una monitorización aproximada al tiempo real del entorno. La red de sensores también debe ser segura, de manera que nadie pueda recopilar datos ajenos o introducirse en la red haciéndose pasar por uno de los nodos de la misma.

Las necesidades del sistema para cumplir con los requisitos técnicos descritos quedan recogidas en la *Tabla 4.1*.

Requisitos	Necesidades
Nodos portables	Un sistema embebido, resistente, que pueda ser introducido en un compartimento estanco de dimensiones reducidas.
Nodos autónomos	En la medida de lo posible con su software debe ser capaz de gestionar el consumo de energía y posibles errores en los mensajes recibidos o el protocolo de comunicación establecido.
Nodos autosuficientes	Debe disponer de un sistema de alimentación propio como una batería y (en una versión más desarrollada) de su propia fuente de energía, como por ejemplo solar.
Nodos capaces de transmitir a grandes distancias	Los módulos de transmisión inalámbrica a emplear deben abarcar varias decenas e incluso cientos de metros para asegurar que pueden conectar con el coordinador de la red o, al menos, con otros nodos cercanos.
Red escalable	Las comunicaciones deben colapsar lo menos posible el medio, por lo tanto el protocolo de comunicación de acceso a la red y validación no debe ser excesivamente largo o complejo. El enrutado no puede incluir tablas de rutado o elementos que impliquen consumo de memoria ya que los nodos disponen de una cantidad limitada de la misma que también se usa como RAM. El sistema también debe permitir que varios coordinadores envíen datos al servidor, es decir, que la validación se debe realizar contra el servidor para evitar que un coordinador desconozca la existencia de ciertos nodos o estos se registren por duplicado.
Comunicaciones seguras	Las comunicaciones entre nodos y el coordinador de la red deben ser cifradas para evitar intrusiones en la red.
Comunicaciones en tiempo real	Las comunicaciones se deben realizar en unos tiempos determinados. No obstante, dada la naturaleza de las comunicaciones inalámbricas que se ven afectadas por el entorno y las condiciones climatológicas, no se puede asumir que la comunicación entre dos nodos siempre se producirá en la ventana de tiempo especificada. Por ello, los nodos deben ser capaces de reiniciar un procedimiento si no reciben respuesta, bien reenviando el mensaje o bien limitando el número de intentos para evitar bloquearse.

Tabla 4.1: Requisitos técnicos

4.3 Requisitos funcionales

El sistema presenta dos partes diferenciadas. Por un lado, la infraestructura de red formada por los nodos y el coordinador y, por otra, la página web que permite administrar el sistema y consultar los datos recopilados. Ambas partes se unen en un punto común que es la base de datos.

En primer lugar se describen los requisitos funcionales de la infraestructura de la red:

- **Solicitar acceso a la red:** los nodos deben ser capaces de presentarse y solicitar acceso al coordinador de la red y, por tanto, a la red, la primera vez que sean introducidos en el sistema y también de buscar un nuevo coordinador si al acceso al anterior falla.

- **Enviar datos al coordinador:** los nodos deben recopilar datos a través de sus sensores y enviarlos al coordinador.

- **Enrutar mensajes:** el sistema en su conjunto debe ser capaz de enrutar mensajes, desde un nodo al coordinador o viceversa, cuando uno de estos no sea capaz de comunicarse con el coordinador de la red por cuestiones de distancia al mismo o por otros elementos tales como obstáculos de gran grosor o interferencias que se lo impidan. Este requisito consta como funcional ya que es necesario implementarlo por software debido a que los módulos de comunicación provistos por la empresa no disponen de esta funcionalidad.

- **Coordinar nodos:** el coordinador debe permanecer a la escucha de los mensajes que le llegan desde los nodos de la red y también debe ser capaz de comunicarse con ellos. De la misma forma, un nodo debe poder convertirse en coordinador para aquellos nodos que no puedan alcanzar al coordinador de la red, siendo a éste al que un nodo vecino envíe todos los mensajes dirigidos al coordinador.

- **Validar nodos:** el coordinador de la red debe poder acceder a la base de datos para verificar que un nodo que solicita acceso a la red está realmente registrado en el sistema y no ha sido validado con anterioridad.

- **Recibir datos de nodos:** el coordinador debe recibir los datos provenientes de los nodos y almacenarlos temporalmente en un fichero.

- **Enviar datos al servidor:** el coordinador debe enviar cada cierto tiempo los datos almacenados al servidor insertándolos en la base de datos.

En segundo lugar se describen los requisitos funcionales del *frontend* web:

- **Consultar datos:** la página web debe permitir visualizar los datos y en la medida de lo posible permitir visualizarlos de forma textual o gráficamente.

- **Administrar nodos:** se debe poder registrar, modificar o eliminar nodos en la base de datos para que puedan ser verificados por el coordinador.

- **Administrar alertas:** debe ser posible registrar, modificar y borrar alertas para monitorizar eventos poco frecuentes o que se pretendan analizar específicamente. Dichas alertas, cuando se desencadene el evento configurado, deben avisar al usuario o usuarios que hayan sido especificados en su creación para que puedan tomar acción.

- **Administrar usuarios:** también debe ser posible registrar usuarios que puedan gestionar nodos y alertas a través del *frontend* web.

4.3.1 Diagrama de casos de uso

De los anteriores requisitos funcionales se deriva el conjunto de casos de uso de la *Figura 4.1.* En dicha figura se representan los diferentes actores del sistema y las distintas interacciones que llevan a cabo.

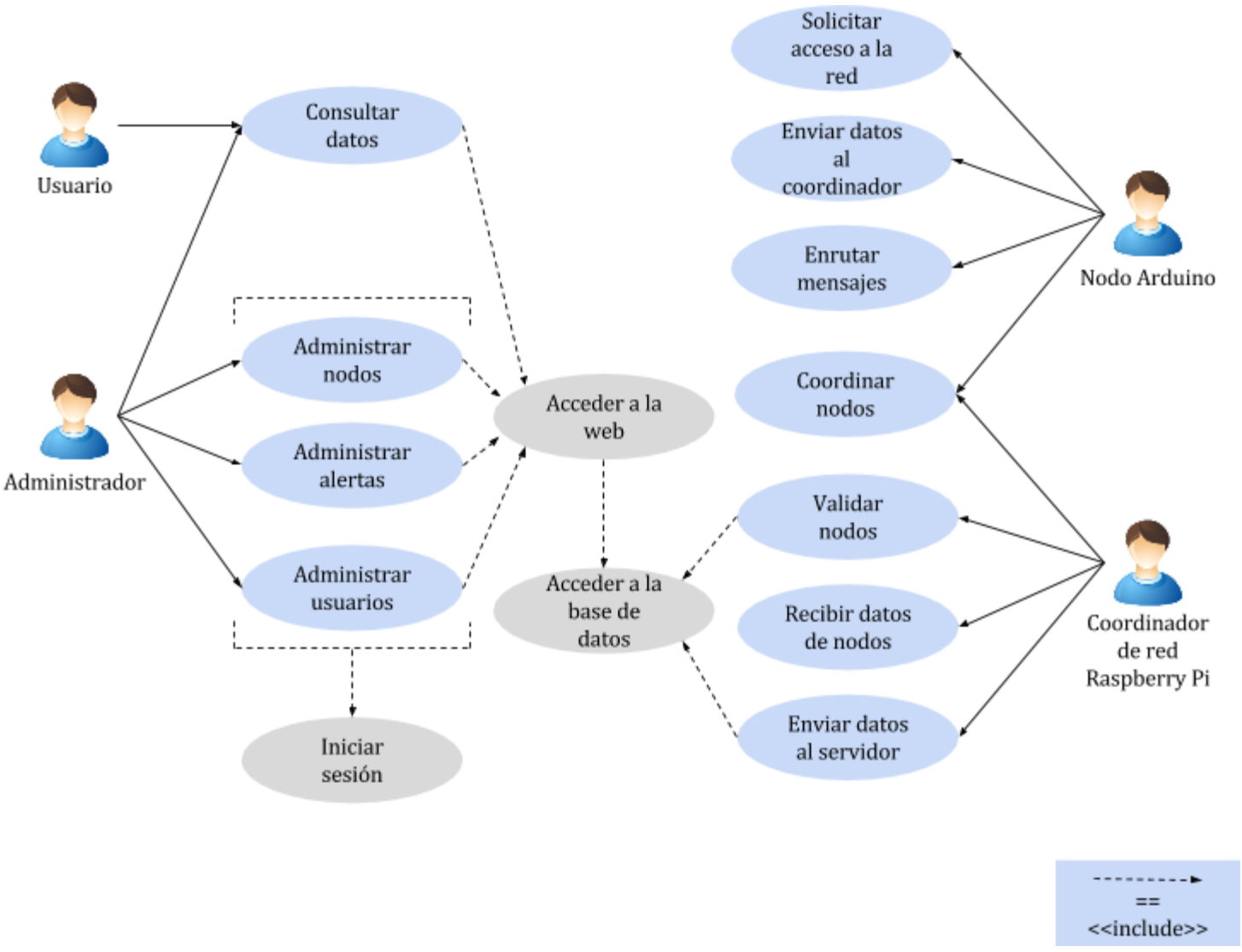


Figura 4.1: Diagrama de casos de uso de los requisitos funcionales

A continuación, se procede a describir los casos de uso para los requisitos funcionales de la infraestructura de la red:

Validar nodos

Origen de la acción:
Coordinador de la red

Destino de la acción:
Nodo Arduino

Descripción:
El coordinador recibe un mensaje de un nodo que quiere usarlo como coordinador.

Escenario:
- El coordinador consulta en la base de datos si el nodo existe y si ha sido validado.
- El coordinador envía un mensaje de confirmación.

Recibir datos de nodos

Origen de la acción:
Nodo Arduino

Destino de la acción:
Coordinador de la red

Descripción:
El coordinador recibe un mensaje con datos de sensores de un nodo y los almacena.

Escenario:
- El coordinador extrae los datos del mensaje.
- El coordinador almacena los datos en un fichero.

Enviar datos al servidor

Origen de la acción:
Coordinador de la red

Destino de la acción:
Servidor

Descripción:
El coordinador envía los datos al servidor cada minuto.

Escenario:
- El coordinador transmite los datos almacenados hasta el momento al servidor.

Solicitar acceso a la red

Origen de la acción:
Nodo Arduino

Destino de la acción:
Nodos vecinos / coordinador de la red

Descripción:
El nodo solicita unirse a la red para transmitir datos de sus sensores.

Escenario:
- El nodo envía un broadcast solicitando datos del coordinador.
- El nodo recibe datos de varios nodos vecinos o del coordinador.
- El nodo solicita al vecino escogido que sea su coordinador.
- El nodo recibe confirmación.
- El nodo pasa a estado activo.

Enviar datos de sensores

Origen de la acción:
Nodo Arduino

Destino de la acción:
Coordinador de la red

Descripción:
El nodo transmite los datos que ha leído de sus sensores al coordinador.

Escenario:
- El nodo lee sus sensores.
- El nodo transmite un mensaje con los datos de los sensores a su coordinador.

Enrutar mensajes

Origen de la acción:
Nodo Arduino / coordinador de la red

Destino de la acción:
Nodo Arduino / coordinador de la red

Descripción:
Un nodo recibe un mensaje de un nodo vecino para que sea transmitido a su coordinador.

Escenario:
- El nodo desempaqueta el mensaje recibido y comprueba su destinatario.
- El nodo reenvía una confirmación al nodo vecino que solicitaba ser enrutado.

Coordinar nodos

Origen de la acción:
Nodo Arduino / Coordinador de la red

Destino de la acción:
Coordinador de la red

Descripción:
Un nodo recibe un mensaje de un nodo vecino para que se convierta en su coordinador ya que no tiene acceso directo al coordinador de la red.

Escenario:
- El nodo recibe una petición para conocer sus datos de acceso al coordinador.
- El nodo envía la información.
- El nodo recibe un mensaje solicitando convertirse en coordinador de otro nodo.
- El nodo consulta al coordinador de la red si se trata de un nodo válido.
- El nodo recibe confirmación del coordinador de la red.
- El nodo envía confirmación al nodo vecino que solicitaba ser rutado.

A continuación, se describen los casos de uso para los requisitos funcionales del frontend:

Consultar datos

Origen de la acción:
Usuario / administrador

Destino de la acción:
Página web

Descripción:
Un usuario accede a la página web para consultar los datos de los sensores.

Escenario:
- El usuario accede a la sección de la página web que permite visualizar datos.

Administrar alertas

Origen de la acción:
Administrador

Destino de la acción:
Página web

Descripción:
El administrador crea, elimina o modifica alertas del sistema.

Escenario:
- El administrador se identifica como tal iniciando sesión.
- Accede a la sección de alertas.
- Crea, modifica o elimina una alerta.

Administrar usuarios

Origen de la acción:
Administrador

Destino de la acción:
Usuario

Descripción:
El administrador crea, elimina o modifica usuarios del sistema.

Escenario:
- El administrador se identifica como tal iniciando sesión.
- Accede a la sección de usuarios.
- Crea, modifica o elimina a un usuario.
- Se envía una notificación por email al usuario.

Administrar nodos

Origen de la acción:
Administrador

Destino de la acción:
Página web

Descripción:
El administrador crea, elimina o modifica registros de nodos del sistema.

Escenario:
- El administrador se identifica como tal iniciando sesión.
- Accede a la sección de nodos.
- Crea, modifica o elimina el registro de un nodo.

4.4 Requisitos de datos

Los requisitos descritos en los apartados anteriores necesitan apoyarse en una serie de estructuras de datos para cumplir con su función. En el caso de MongoDB no hablamos de tablas sino de colecciones. A continuación, se introducen las distintas necesidades de almacenamiento de datos y se indican los campos de la colección creada para cubrirla.

Los nodos deben transmitir la información de los sensores que tengan instalados al coordinador de la red y éste a su vez debe redirigirlos a la base de datos en el servidor. En la *Tabla 4.2* se detallan los campos que contienen la información almacenada de los sensores:

Colección 1: datos	
Datos recogidos	● Identificador del nodo origen (dirección 64 bits) ● Identificador del sensor del que provienen los datos ● Valor recogido ● Fecha y hora de recepción ● Ubicación y coordenadas

Tabla 4.2: Datos recogidos por los sensores

Por otra parte, para poder enrutar datos desde el coordinador de la red hasta un nodo no alcanzable directamente es necesario que el coordinador conozca a través de qué nodos debe enviar un paquete. En la *Tabla 4.3* se indican los campos utilizados para almacenar las rutas.

Colección 2: rutas	
Datos de enrutado	● Identificador del nodo (dirección 64 bits) ● Identificador de nodo enrutador (dirección 64 bits)

Tabla 4.3: Datos de enrutado

También se debe almacenar información referente a los nodos para que puedan ser identificados como pertenecientes a la red. En la *Tabla 4.4* se detallan los campos del registro de la base de datos que contienen dicha información.

Colección 3: nodos	
Datos de nodos	● Identificador del nodo (dirección 64 bits) ● Fecha de registro en la base de datos ● Estado en la red (validado / no validado) ● Fecha y hora de validación ● Coordinador / enrutador (dirección 64 bits) ● Descripción ● Ubicación ● Coordenadas ● Habilitado (sí / no)

Tabla 4.4: Datos de los nodos

Los nodos disponen de un conjunto de sensores que también son recogidos en la base de datos para mantener un registro de sus características, tal y como puede verse en la *Tabla 4.5*.

Colección 4: sensores	
Datos de sensores	● Identificador del sensor ● Nombre técnico del componente ● Descripción del tipo de medición ● Unidad en la que se realizan las mediciones (p.ej.: °C) ● URL a información adicional (p.ej. un datasheet) ● Fecha de creación ● Habilitado (sí / no)

Tabla 4.5: Datos de sensores

Para la gestión de las alertas también se debe incluir un registro en el que se especifique la configuración de una alerta y la acción a tomar. En la *Tabla 4.6* se muestra la información almacenada para la configuración de las alertas.

Colección 5: alertas	
Datos de alertas	● Configuración de la alerta (cadena formateada) ● Fecha de inicio de la validez de la alerta ● Fecha de fin de validez de la alerta ● Acción ● Usuarios involucrados ● Fecha de creación de la alerta ● Usuario que creó la alerta ● Descripción ● Habilitada (sí / no)

Tabla 4.6: Datos de alertas

Por último, en la *Tabla 4.7* se detallan los campos del registro de los usuarios que tienen acceso al sitio web para gestionar usuarios, alertas y/o sensores.

Colección 6: usuarios	
Datos de usuarios	● Nombre ● Apellidos ● Fecha de registro ● Fecha de baja (equivalente a habilitado) ● Nombre de usuario ● Contraseña ● Email ● Teléfono ● Permisos de gestión (15 bits -> datos, usuarios, nodos, alertas, sensores)

Tabla 4.7: Datos de usuarios

Capítulo 5

Diseño de la arquitectura del sistema

En este capítulo se describe el diseño de la arquitectura del sistema, que se divide en dos vertientes:

- La interacción de los componentes entre sí como entes concretos representados por su propio hardware y software, y que es descrita en el apartado 5.1.
- La interacción de los elementos software que definen su comportamiento y que son desglosados para cada uno de los componentes en el apartado 5.2.

También se detalla, en el apartado 5.3, el diseño de la interfaz del *frontend* web.

5.1 Diseño de la arquitectura de la red

Como se ha mencionado en capítulos anteriores, el sistema de monitorización del entorno está compuesto por tres elementos clave: nodos, coordinador de la red y frontend, que se relacionan entre sí a través del servidor y su base de datos. A partir del análisis del sistema se ha definido la arquitectura representada en la *Figura 5.1.*

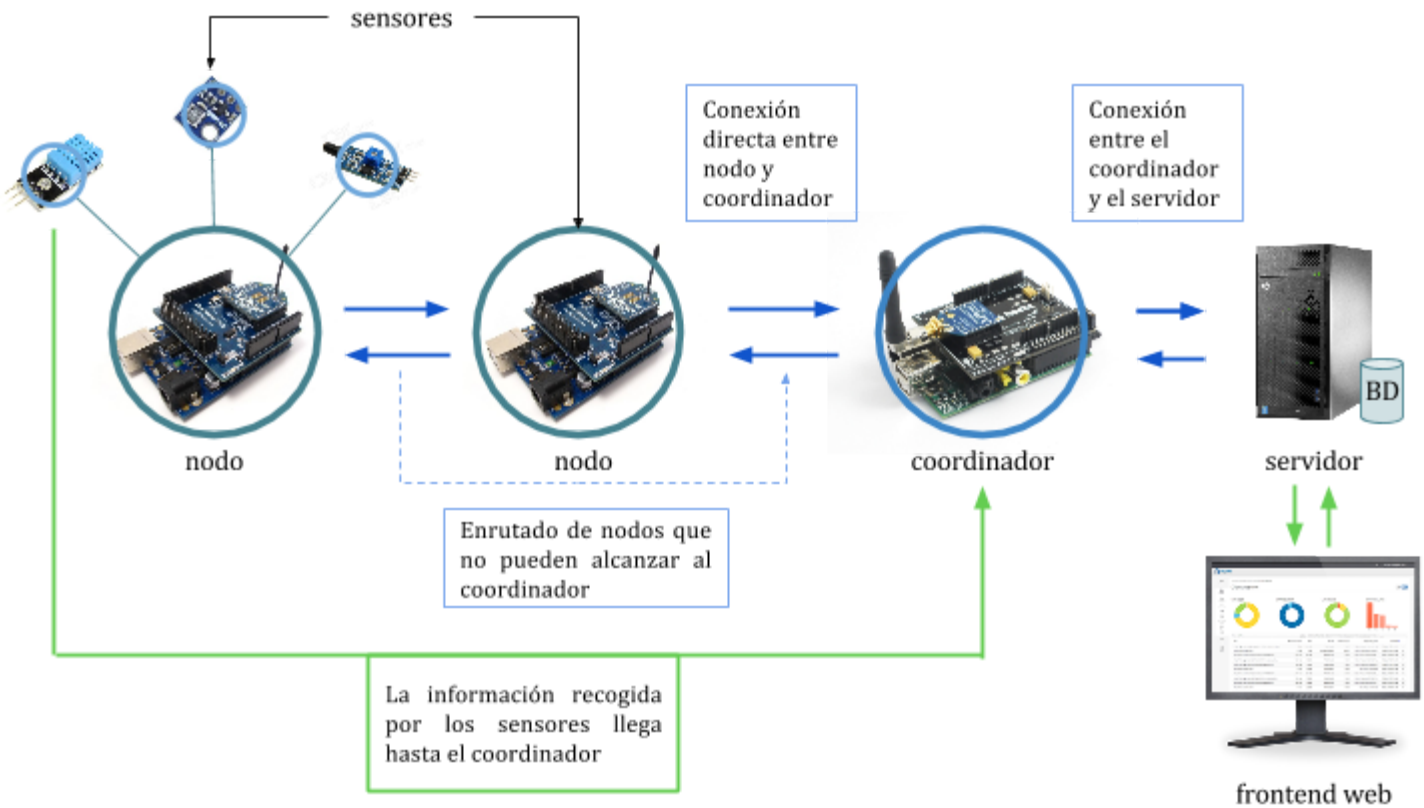


Figura 5.1: Esquema general del sistema de monitorización del entorno

A partir de la *Figura 5.1* se resume el papel de cada uno de estos tres componentes en la red de la siguiente forma:

- Los nodos Arduino son la parte más básica de la red de monitorización ya que son quienes recopilan los datos. Son capaces de:
  - Unirse automáticamente a la red.
  - Comunicarse con el coordinador.
  - Enrutar mensajes de otros nodos hacia el coordinador y viceversa.
  - Transmitir los datos recopilados por los sensores.

- El coordinador de red Raspberry Pi actúa de enlace entre los nodos y el servidor. Como coordinador de red es capaz de:
  - Validar a los nodos que se unan a la red, de forma directa o a través de un nodo que haga las veces de enrutador.
  - Comunicarse con nodos distantes a través de un nodo enrutador.

  Y como puerta de enlace que se comunica con el servidor es capaz de:
  - Realizar consultas en la base de datos.
  - Realizar inserciones remotas en la base de datos para almacenar la información provenientes de los sensores.

- Por último, el *frontend*, como punto de acceso del usuario al sistema, permite:
  - Consultar los datos recabados por los sensores.
  - Realizar modificaciones en la base de datos (gestión de nodos y alertas).

5.1.1 Topología de red

A partir del esquema presentado en la Figura 5.1 se han considerado diversas topologías, las cuales describen la forma en la que los nodos se comunican físicamente entre sí. Encontramos principalmente tres tipos de topologías de red [9], además de la conexión punto a punto, tal y como se puede observar en la *Figura 5.2*.

Como se menciona en el *Capítulo 2*, los módulos XBee disponibles para este proyecto sólo son capaces de comunicarse punto a punto o multipunto mediante el envío de un broadcast y no disponen de capacidad de enrutado por sí solos. Es decir, los nodos por sí mismos sólo podrían tomar el papel de coordinadores, en el sentido de poder recibir mensajes desde cualquier nodo, o de dispositivos finales, siendo configurados para transmitir únicamente al dispositivo coordinador.

Por tanto, todos los nodos han sido configurados como coordinadores, recordemos que esto se refiere a que pueden recibir mensajes desde cualquier dirección, y cualquier funcionalidad de enrutado ha sido implementada por software. Así que, cuando en este documento se habla de nodo coordinador hace referencia a un dispositivo (una Raspberry Pi con un módulo XBee) que realiza esta función dentro de la red y cuando se habla de nodo/nodo final o enrutador hace referencia a un nodo Arduino con sensores y un módulo XBee, tal y como se describe en el *Capítulo 2* y en el esquema de la *Figura 5.1*.

Así pues, a continuación se explican brevemente las topologías mencionadas y se presenta la solución por la que se ha optado en base a las restricciones descritas anteriormente:

- Topología punto a punto: es la más sencilla, los nodos se conectan directamente a otro nodo que hace de coordinador.

- Topología en estrella: el nodo coordinador se sitúa en el "centro" de la topología y el resto de nodos transmiten directamente a este nodo coordinador quién se encarga de enrutar los mensajes hacia otros nodos de la red.

- Topología en malla (o *Mesh*): se utilizan nodos enrutadores que complementan al nodo coordinador. Estos nodos transmiten mensajes entre nodos enrutadores o a los nodos finales

- Topología de árbol de clusters: sigue un esquema similar a la topología en malla. Los nodos transmiten a un nodo enrutador el cual se comunica con otros nodos enrutadores o con el nodo coordinador.

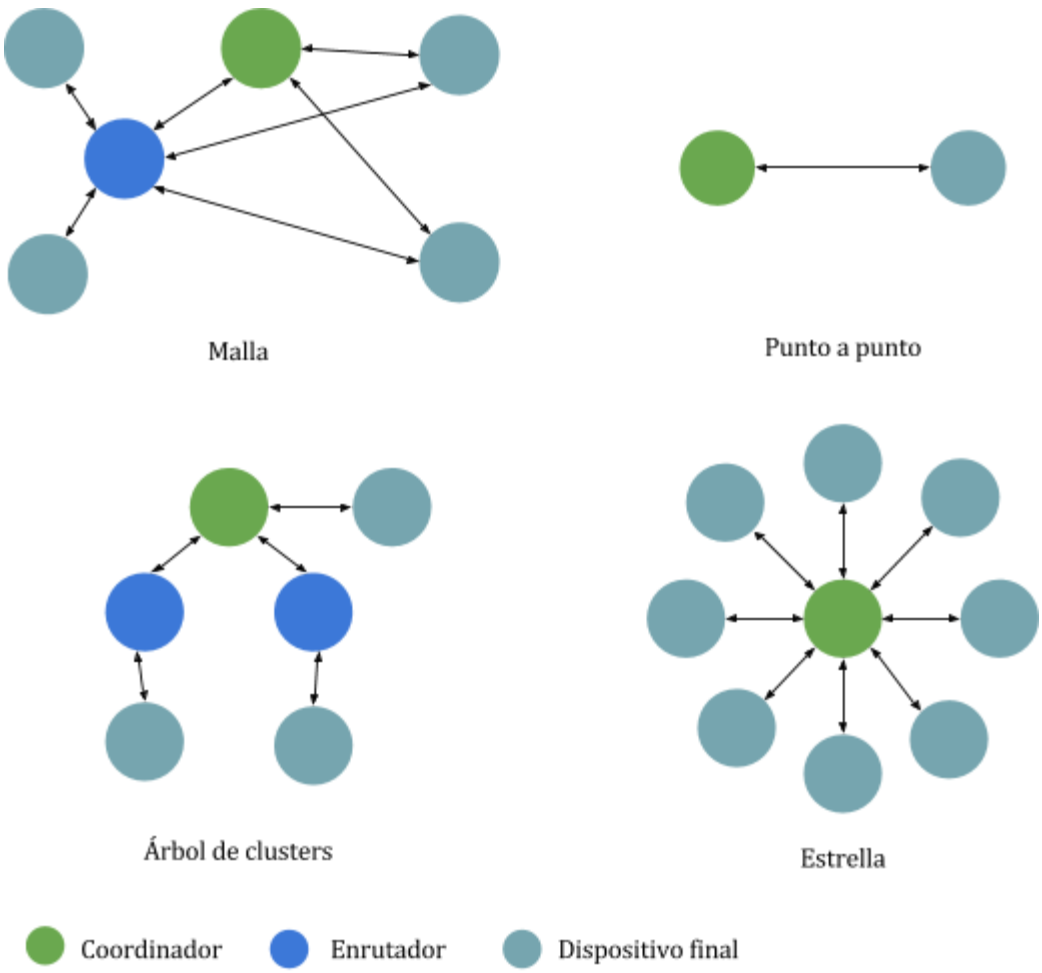


Figura 5.2: Tipos de topologías de red

A continuación se indican las características de la red de los sensores:

- Pueden añadirse nuevos nodos contínuamente.
- Estos nodos no necesariamente presentarán una distribución uniforme en el espacio.
- Los nodos transmiten datos únicamente al coordinador de la red.
- No obstante, hay que tener en cuenta que cuando uno de ellos no pueda acceder al coordinador de la red sus mensajes son enrutados por otros nodos.

Así pues, tras analizar las diferentes topologías se ha determinado que:

- La topología punto a punto queda descartada ya que no considera el enrutado entre nodos.

- La topología en estrella también queda descartada, pues los nodos no necesitan enviarse mensajes entre sí, salvo con los que tienen visión directa si no alcanzan al coordinador, y no da una solución al problema del acceso al coordinador de la red, ya que el resto de nodos no enrutan.

- La topología de árbol de clusters podría ser una buena opción, sin embargo, el hecho que la distribución de los nodos no sea uniforme y además sea variable, puede ocasionar que un nodo enrutador se encuentre excesivamente saturado mientras que otros pasen la mayor parte de su tiempo ociosos. Además, este sistema obligaría a tener nodos específicos para enrutar, lo que supondría una complicación a la hora de añadir nuevos nodos en zonas distantes a las ya establecidas, además de un incremento en el coste del producto.

Por tanto, el tipo de topología que más se adapta al proyecto es la de malla ya que:

- Permite que un nodo de la red haga las funciones de enrutador. Lo que en el proyecto se ha interpretado como un nodo convencional que, en un momento dado, y por petición de otro nodo, se convierte en enrutador pero sin dejar de lado sus funciones de recogida de datos, ya que se limita a retransmitir los datos que le llegan del nodo enrutado al coordinador.

- Los nodos, mientras les sea posible, pueden comunicarse directamente con el nodo coordinador. De este modo se simplifica el protocolo de comunicación y la gestión de los mensajes al existir un único punto de envío que, además, dispone de un hardware más potente y adecuado para procesar la recepción de grandes cantidades de mensajes.

- Esta topología tampoco limita la distribución de los nodos ya que si tienen acceso al coordinador se comunicarán directamente con éste y, sino, un nodo vecino cualquiera se encargará de enrutar sus mensajes.

La representación de la topología definitiva puede verse en la *Figura 5.3*:

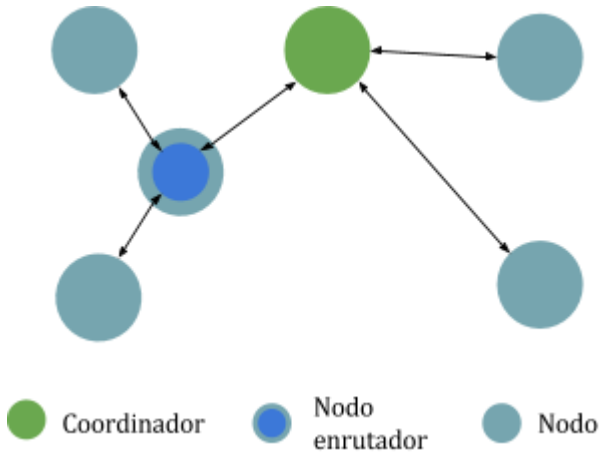


Figura 5.3: Topología de la red desarrollada para el proyecto

5.1.2 Formato de las comunicaciones

Los módulos XBee [3] han sido configurados en el modo API2. Esto significa que los mensajes no son enviados como texto plano al medio sino que siguen un formato determinado y que los caracteres son escapados para evitar que algunos sean interpretados como caracteres de control durante la recepción de un mensaje. Los caracteres escapados son: 0x7E que se corresponde con el inicio de una trama, 0x7D que indica que el carácter que sigue a continuación ha sido escapado y 0x11 y 0x13 que son caracteres de control de flujo.

Escapar un carácter, durante el envío de un mensaje, significa enviar el carácter 0x7D y a continuación el carácter a enviar aplicándole la operación lógica XOR con 0x20. La acción contraria, cuando se recibe un mensaje, consiste en ignorar el caracter 0x7D y realizar la operación lógica mencionada con el carácter que sigue a continuación [10].

El formato general de un mensaje en el modo API2, tal y como se observa en la *Figura 5.4*, es el siguiente:

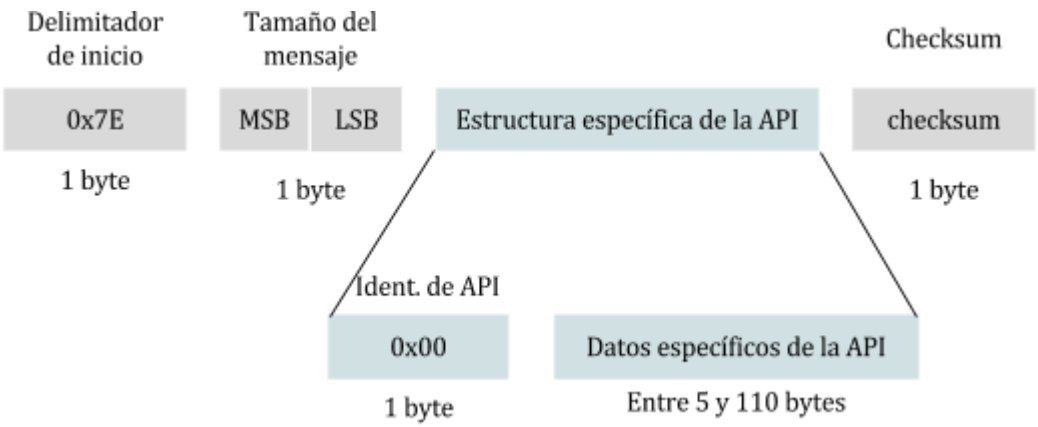


Figura 5.4: Formato general de un mensaje en el modo API2

- El delimitador de inicio marca el inicio de una trama entrante. Su valor es siempre: 0x7E.

- El tamaño del mensaje incluye todos los campos. En este proyecto los mensajes nunca ocupan más de 115 bytes, es decir que la información cabe en una sola trama y se envía siempre de una sola vez. Así pues, el primer byte de los dos que definen el tamaño de los datos siempre será 0x00 y el segundo indicará el tamaño total, ya que con 8 bits se puede representar números hasta el 255.

- La estructura específica de la API. Define el tipo de mensaje y contiene la información que se desea transmitir. Se explica en detalle en el siguiente subapartado.

- El checksum es la suma de los valores de la estructura específica de la API y se utiliza para comprobar si el mensaje ha llegado correctamente [11].

Estructura específica de la API

La biblioteca XBee desarrollada para el proyecto, y que se describe más adelante, implementa el tratamiento de las cuatro API básicas definidas en el protocolo de los módulos XBee Serie 1:

- API 0x08: envío de un comando de configuración AT local (ver *Figura 5.5)*.
- API 0x88: respuesta de un comando de configuración AT local (ver *Figura 5.5)*.
- API 0x00 y 0x01: envío a una dirección de 64 ó 16 bits (ver *Figura 5.6)*.
- API 0x80 y 0x81: recepción desde una dirección de 64 ó 16 bits (ver *Figura 5.6)*.

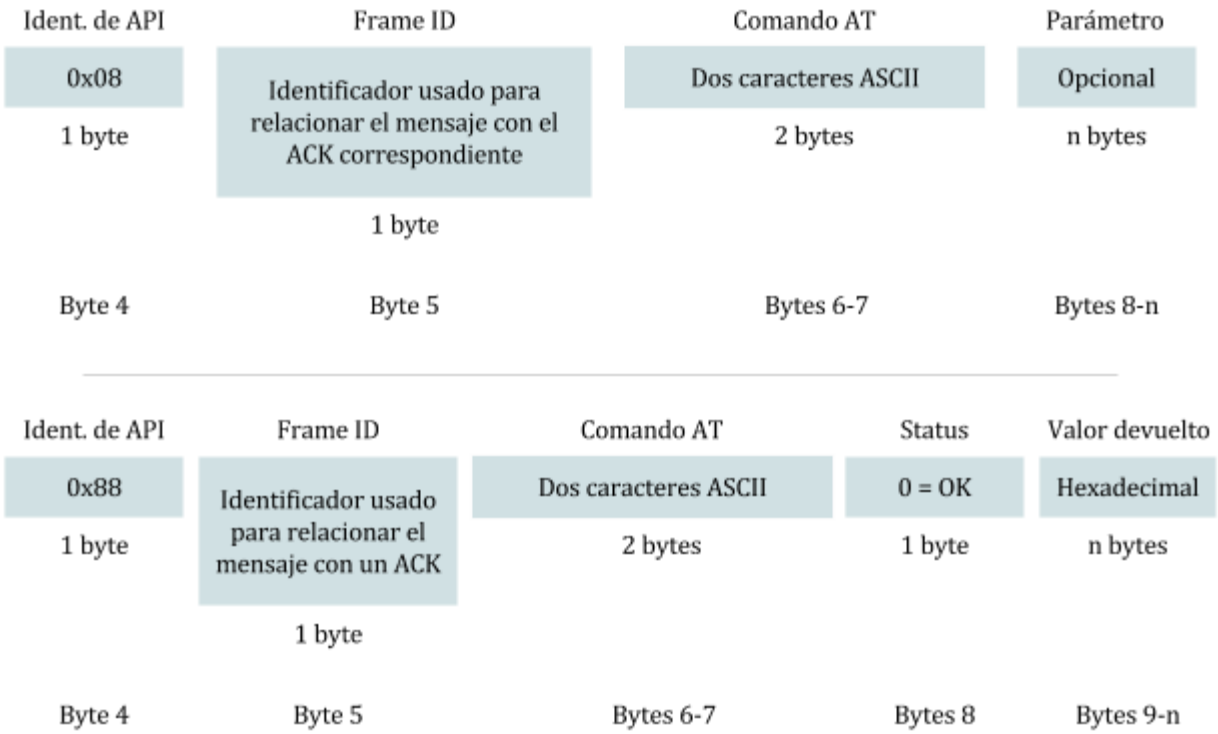


Figura 5.5: API de envío (superior) y respuesta (inferior) de un comando de configuración AT local

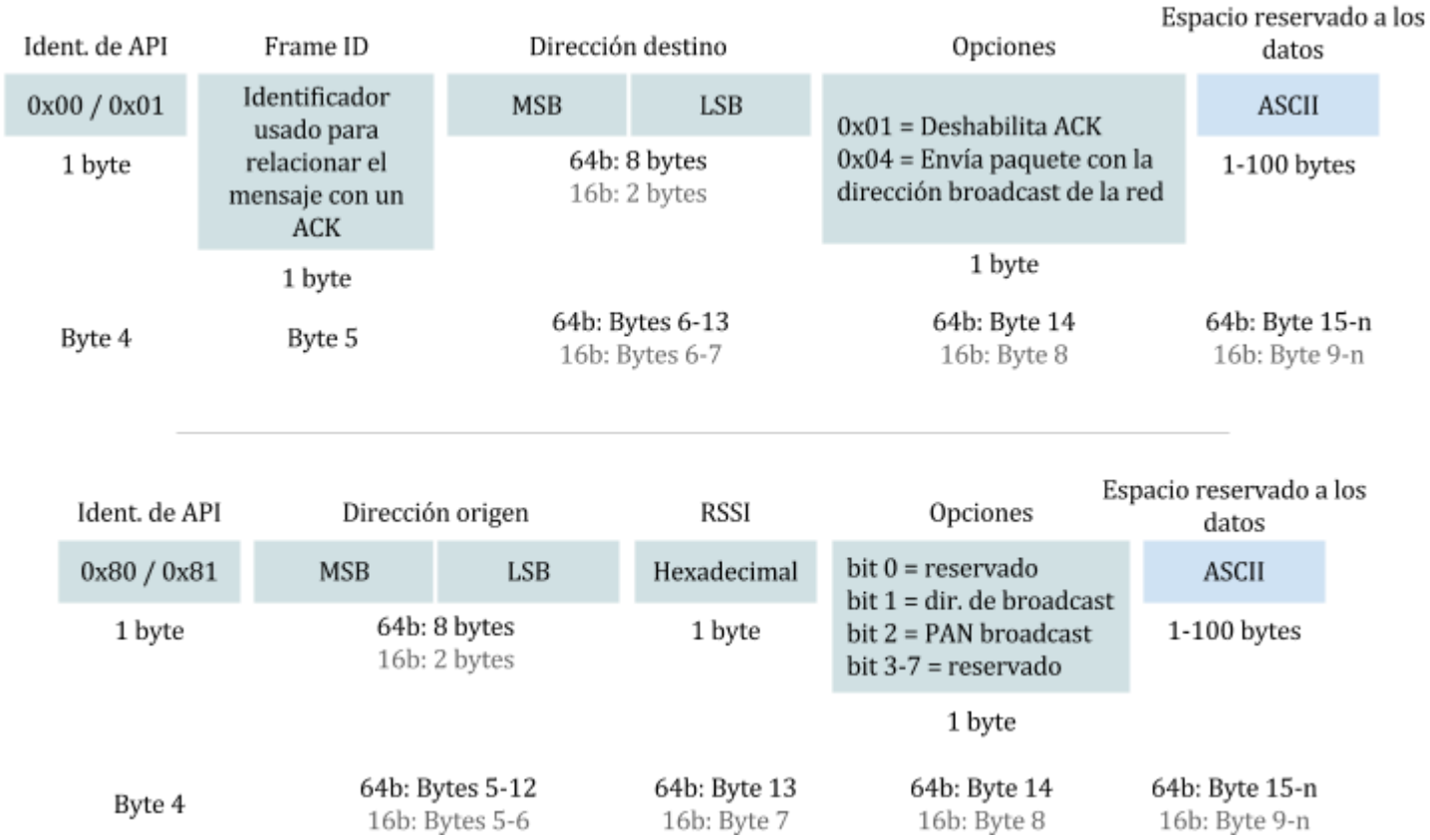


Figura 5.6: API de envío (superior) y recepción (inferior) de datos para direcciones de 64 y 16 bits

Además, para identificar el tipo de mensaje, se ha implementado un sistema de cabeceras específico para el proyecto. Se ha determinado que en el campo destinado a los datos, el primer byte (0) se corresponderá con el identificador del tipo de mensaje o cabecera. De esta forma se agiliza el tratamiento de los mensajes.

Se han definido nueve cabeceras distintas con su correspondiente estructura. A continuación se describen cada una de ellas:

- Cabecera 0x31: mensaje enrutado destinado al coordinador de la red. Su estructura es la siguiente:

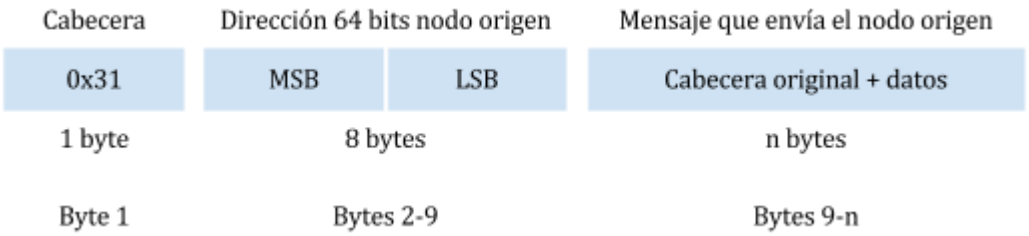


Figura 5.7: Cabecera 0x31

- Cabecera 0x32: mensaje que solicita al coordinador que confirme un nodo que quiere unirse a la red a través de otro nodo. Su estructura es la siguiente:

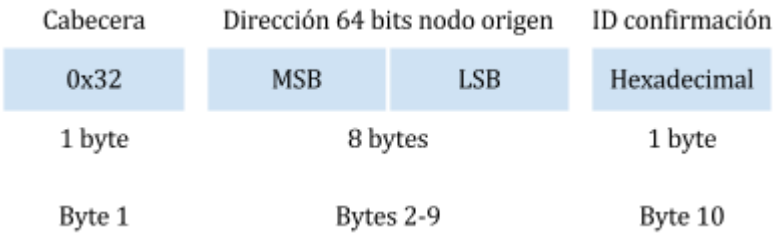


Figura 5.8: Cabecera 0x32

Algunas cabeceras incluyen un campo denominado "ID confirmación", este campo contiene un identificador que el emisor del mensaje guarda y que el destinatario del mensaje incluye en su respuesta a modo de ACK. Es la forma en la que el emisor relaciona una petición y su respuesta.

- Cabecera 0x33: mensaje enviado por el coordinador a un nodo al que no tiene acceso directo. Su estructura es la siguiente:

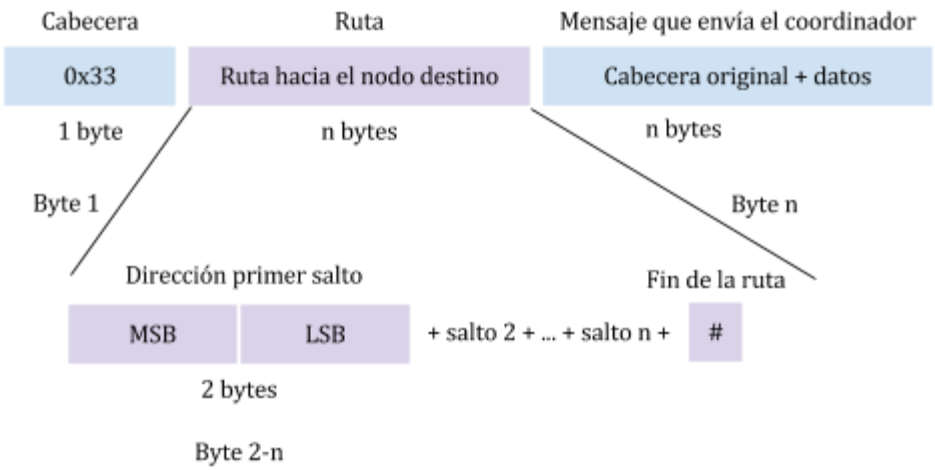


Figura 5.9: Cabecera 0x33

La distancia a la que podrá enviarse la trama, o número de saltos, dependerá del tamaño del mensaje a enviar. Se calcula como:

$$\text{Nº saltos} = \frac{100 - cabecera\_trama - caracter\_fin\_ruta - tamaño\_mensaje}{tamaño\_direccion\_nodo\ (16 \text{ ó } 64\ bits)}$$

Siendo 100 el tamaño total de bytes de datos que pueden transmitirse en un solo frame y *caracter_fin_ruta* el carácter '#' que puede observarse en el último campo del campo "Ruta" de la *Figura 5.9*.

- Cabecera 0x34: a modo de ACK, se utiliza como confirmación/denegación de un procedimiento (por ejemplo de la verificación de un nodo). Su estructura es la siguiente:

Cabecera	ID confirmación	Status
0x34	Hexadecimal	1 = OK
1 byte	1 byte	1 byte
Byte 1	Byte 2	Byte 3

Figura 5.10: Cabecera 0x34

- Cabecera 0x35: mensaje de un nodo solicitando los datos de conexión al coordinador de un vecino. Su estructura, que incluye únicamente un campo con la cabecera, es la siguiente:

Cabecera
0x35
1 byte
Byte 1

Figura 5.11: Cabecera 0x35

- Cabecera 0x36: mensaje con los datos de conexión al coordinador de un nodo vecino. Su estructura es la siguiente:

Cabecera	RSSI con el coord.	Directo al coord.	Coord. del nodo tiene visión directa al coord. de la red
0x36	Hexadecimal	1 = Sí	1 = Sí
1 byte	1 byte	1 byte	1 byte
Byte 1	Byte 2	Byte 3	Byte 4

Figura 5.12: Cabecera 0x36

- Cabecera 0x37: mensaje indicando que el nodo ha sido escogido como coordinador por otro nodo vecino que solicitó sus datos con anterioridad. Su estructura es la siguiente:

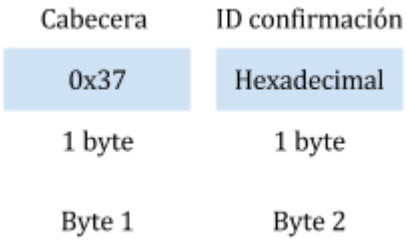


Figura 5.13: Cabecera 0x37

- Cabecera 0x38: un nodo vecino pregunta al nodo si sigue activo.

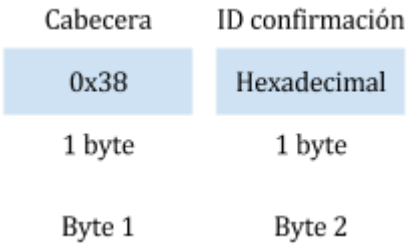


Figura 5.14: Cabecera 0x38

- Cabecera 0x39: mensaje con datos de los sensores. Su estructura es la siguiente:

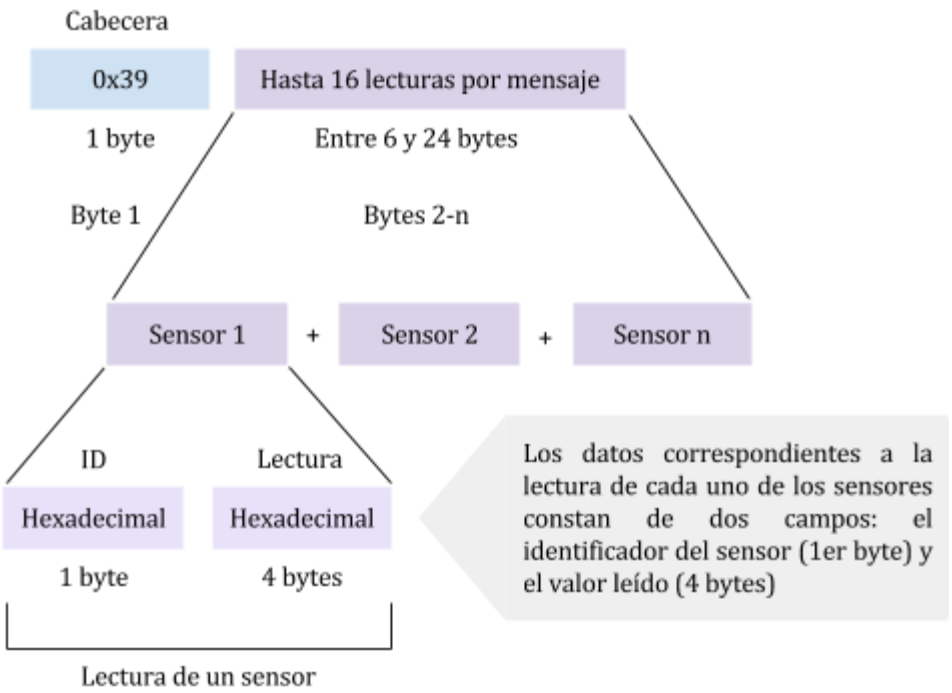


Figura 5.15: Cabecera 0x39

5.1.3 Direccionamiento y enrutado

Los nodos sólo se comunican enviando sus mensajes directamente al coordinador, es decir, realizando transmisiones punto a punto. Si no pueden acceder directamente al coordinador de la red transmiten sus mensajes al nodo vecino que hayan designado como su coordinador. Para el direccionamiento entre nodos es posible utilizar direcciones de 16 y 64 bits. Las primeras están pensadas para redes con hasta 65536 nodos mientras que las de 64 bits lo están para construir redes virtualmente ilimitadas con hasta 2^64 elementos. Además, estas direcciones se corresponden con la MAC del dispositivo y por tanto son únicas. Por ello, para este proyecto se ha decidido trabajar con direcciones de 64 bits.

A continuación, se detallan los procesos de unión de un nodo a la red y enrutado.

Unión de un nodo a la red

Al unirse a la red, los nodos deben iniciar la búsqueda del coordinador enviando un broadcast. Todos los nodos al alcance con acceso al coordinador, incluido el propio coordinador de la red, cuando reciban este mensaje responderán enviando los datos de su conexión al nodo. El nodo solicitante por su parte comparará las respuestas de los otros nodos y almacenará los datos del mejor coordinador de forma temporal. Tras finalizar esta búsqueda de vecinos, el nodo enviará un mensaje al coordinador escogido solicitando unirse a la red a través de él. Si se trata de un nodo enrutador, es decir que no es el coordinador de la red, éste notificará al coordinador la dirección del nodo que desea unirse para que sea comprobado en la base de datos. En caso de que el coordinador escogido sea el de la red realizará la comprobación directamente a la base de datos. Una vez el nodo que ha solicitado unirse a la red reciba la confirmación comenzará a transmitir datos de los sensores. Los nodos enrutados por un nodo vecino además comprobarán cada cierto tiempo que su coordinador sigue "vivo" y de no ser así iniciarán la búsqueda de un nuevo coordinador. La *Figura 5.16* representa de forma gráfica la unión de un nodo a la red.

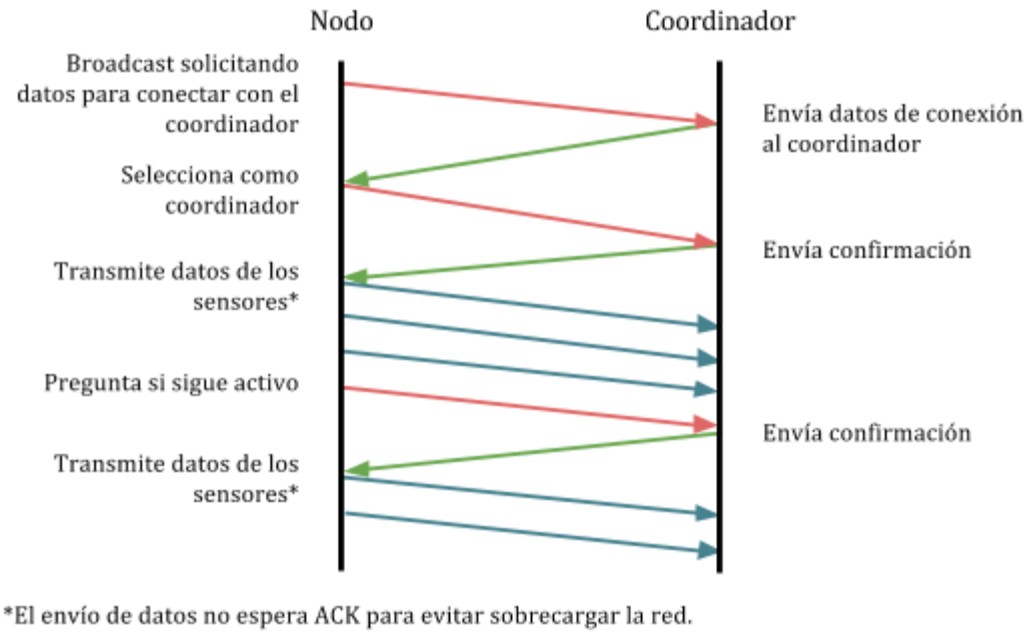


Figura 5.16: Proceso de unión de un nodo a la red

Enrutado de mensajes

Por su parte, el enrutado de mensajes desde un nodo al coordinador consistirá en empaquetar el mensaje original con un encabezado en la sección de datos que indique que el destino es el coordinador de la red. Además, se incluirá tras éste la dirección del nodo origen. Cuando un nodo enrutador reciba uno de estos mensajes se limitará a retransmitirlo a su coordinador. Este comportamiento se puede observar en la *Figura 5.17*.

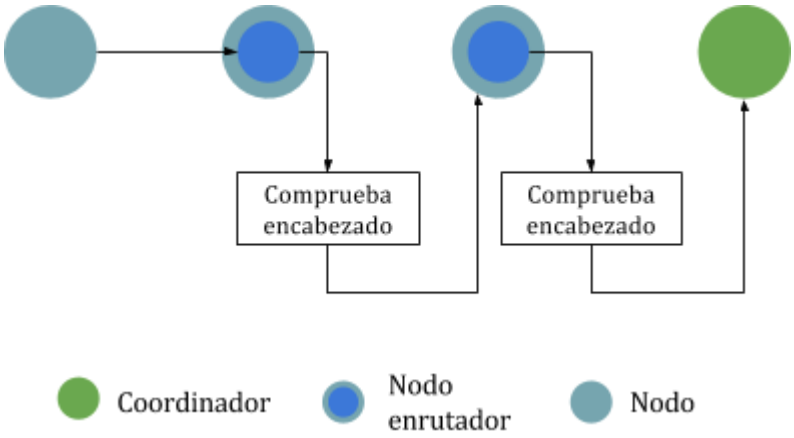


Figura 5.17: Enrutado desde un nodo al coordinador

Si el enrutado se produce desde el coordinador a un nodo distante se seguirá un procedimiento similar. En primer lugar, el coordinador consultará en la colección de la base de datos que alberga las rutas entre nodos quién es el nodo con el que debe comunicarse. Si la entrada correspondiente al nodo enrutador está vacía, la comunicación con el nodo será directa. Por contra, si aparece la dirección de otro nodo, se deberá acceder a su correspondiente entrada en la base de datos para comprobar si el acceso es directo o través de otro nodo. Tras localizar al primer nodo al que el coordinador tenga acceso directo, se conformará una ruta que se añadirá al mensaje a enviar, tal y como se muestra en la *Figura 5.9*. Cuando uno de los nodos enrutadores de dicha ruta reciba el mensaje, lo desempaquetará y enviará al siguiente destinatario. Este procedimiento se puede observar en la *Figura 5.18*.

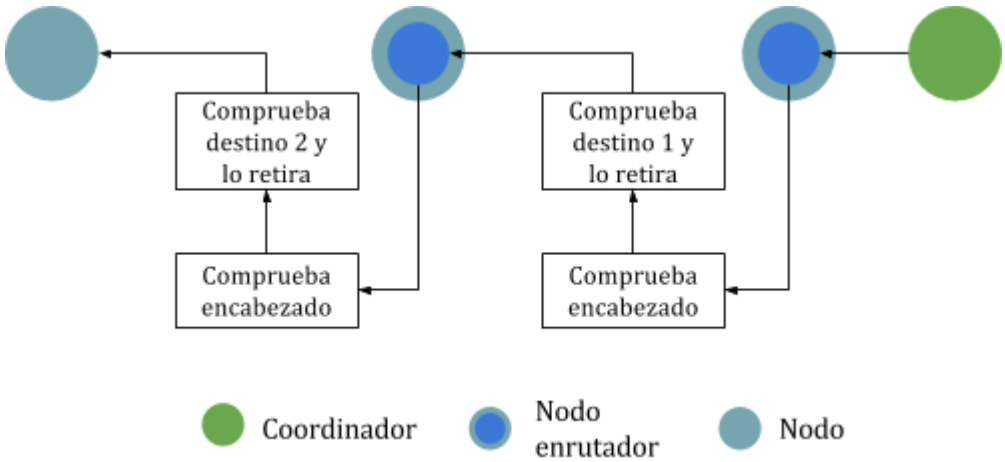


Figura 5.18: Enrutado desde el coordinador a un nodo

5.1.4 Seguridad

Actualmente la seguridad en las transmisiones que se realizan en la red de monitorización viene dada por la capa de control de acceso al medio definida por el protocolo 802.15.4 [4] utilizado por los módulos XBee. Este protocolo proporciona cuatro frentes de seguridad:

- Control de acceso con el que el módulo mantiene una lista de los módulos validados.
- Datos cifrados con AES [14] utilizando una clave de 128 bits.
- Integración de tramas para evitar que la información transmitida sea alterada.
- Secuencias de refresco para comprobar que las tramas no han sido reemplazadas.

Todos los módulos XBee de una red son configurados con una misma clave de enlace para el cifrado de las comunicaciones. Si un nodo dispone de esta clave de enlace, cualquier nodo configurado como coordinador podrá compartir con él la clave de red para que pueda comunicarse con el resto de nodos.

Otro aspecto que contribuye a la seguridad de la red es que un nodo que no conste en la base de datos no será enrutado por ningún otro, ni sus mensajes aceptados por el coordinador. De esta forma ante un posible robo, malfuncionamiento o pérdida de un nodo activo, éste puede ser dado de baja en la red, previniendo de esta forma que datos erróneos o alterados sean incorporados a la base de datos.

5.1.5 Gestión de la red

Toda la gestión de la red es llevada a cabo por el coordinador de red. Aunque es posible que varios coordinadores trabajen juntos en una misma red, ya que la validación de los nodos y la consulta de las rutas para nodos distantes se apoya en una misma base de datos en el servidor.

Esta gestión incluye que el coordinador actualice la lista de nodos validados que posee para comprobar posibles exclusiones o incorporaciones a la red por los administradores, así como la lista de las rutas hacia los nodos inalcanzables ya que estas pueden crecer, cambiar o desaparecer.

Respecto a estas dos estructuras, la lista de rutas y de nodos, mientras que la primera es construida por el propio coordinador, la administración de los nodos es llevada a cabo por los usuarios mediante el *frontend* web.

5.2 Diseño de la arquitectura lógica

Como se indicó en el *Capítulo 2*, el software que controla las acciones de los nodos y del coordinador está formado por un programa principal y la biblioteca XBee, que junto a la biblioteca SerialManager proporcionan funciones básicas para el uso de los módulos XBee. También se utilizan otras bibliotecas de soporte como SnoozeLib, QueueList o MongoDB C++ driver.

En la *Figura 5.19* se muestra un esquema de los componentes lógicos de los nodos y el coordinador, en el que se pueden ver los componentes software mencionados y cómo se relacionan entre sí.

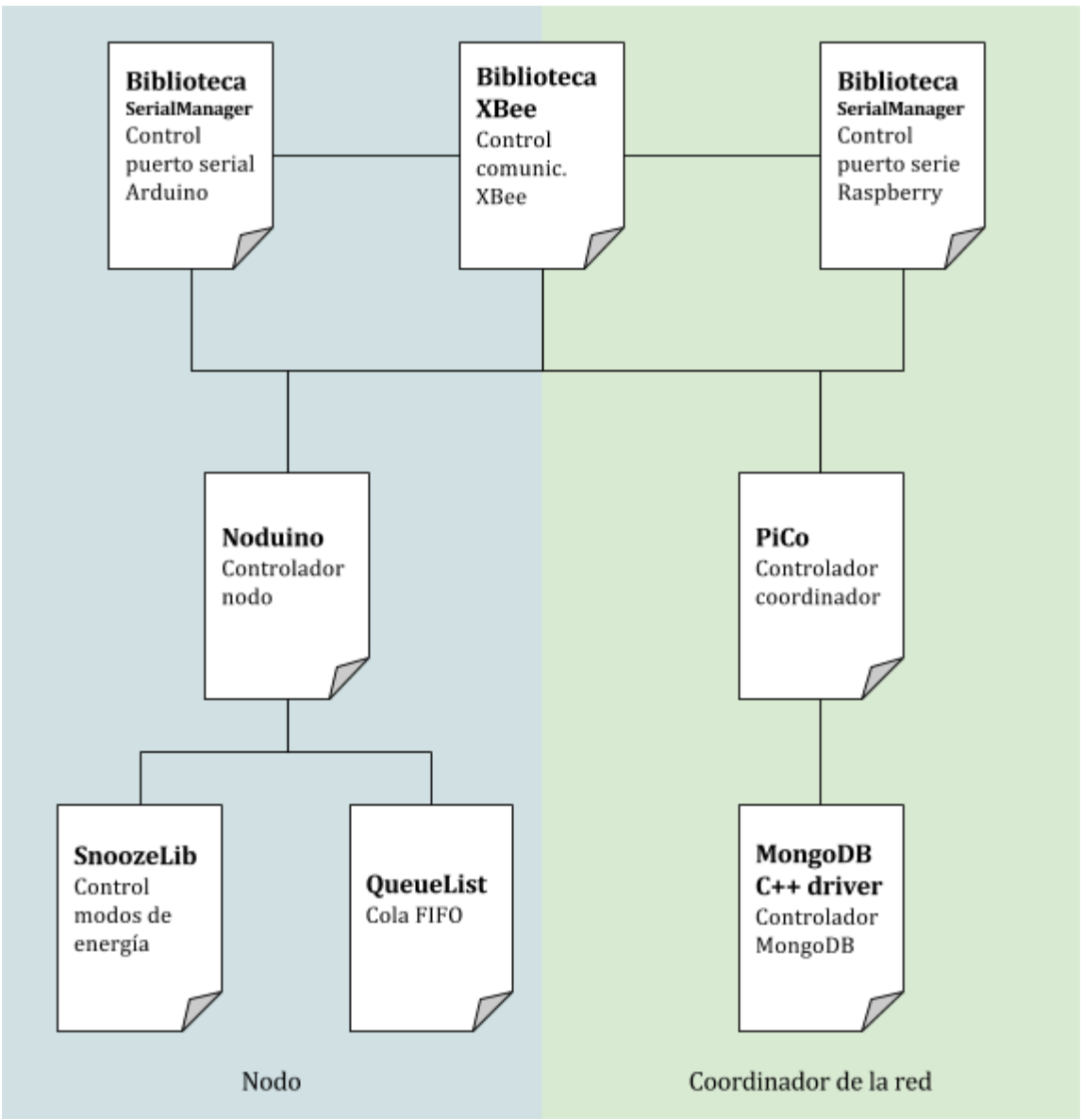


Figura 5.19: Esquema lógico de la red para los nodos y el coordinador

Por otra parte, en el servidor se encuentran la base de datos en MongoDB y el *frontend* web formado por la página web y el gestor de alertas. Al igual que los componentes software de la red, también se apoyan en otras bibliotecas y tecnologías como PHP o los drivers de MongoDB para C++ y PHP. En la *Figura 5.20* se muestra la relación de estos componentes.

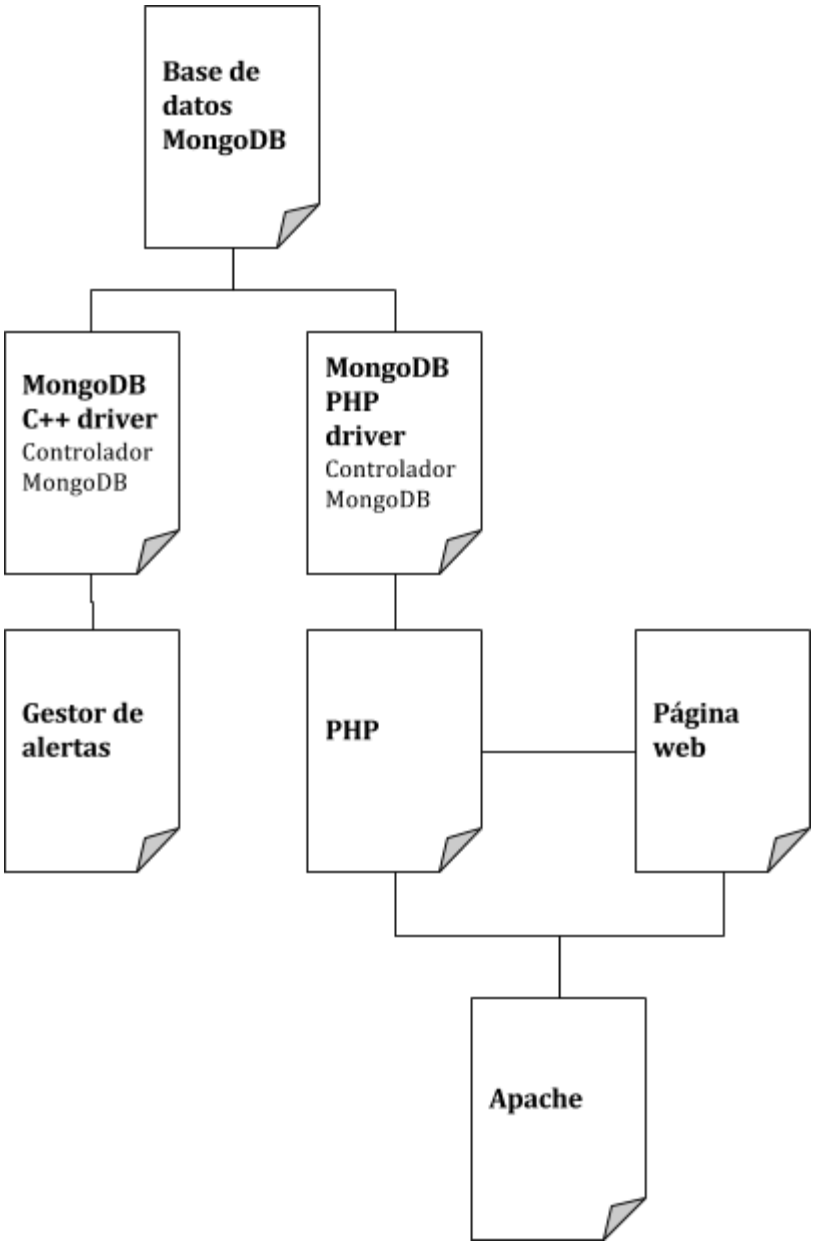


Figura 5.20: Esquema lógico del servidor y el *frontend*

A continuación, se definen cada uno de estos componentes software, los cuales han sido desarrollados expresamente para este proyecto, y la funcionalidad que proporcionan al sistema o a otros elementos de la arquitectura lógica.

5.2.1 Biblioteca XBee y SerialManager

La biblioteca XBee es la que permite a los programas que controlan a los nodos y al coordinador utilizar los módulos XBee para enviar y recibir datos. Esta librería es compartida por los nodos y el coordinador de la red, ya que ambos emplean los mismos módulos de transmisión inalámbrica.

Dispone de una función para autoconfigurar los módulos XBee mediante los comandos AT locales descritos en el apartado *5.1.2*, *Figura 5.5*. Los parámetros configurados son:

- Modo API a 2 (comando *ATAP 2*): para usar mensajes estructurados con escape de caracteres de control.
- Baudaje a 9600 baudios (comando *ATBD 9600*) : representa el número de símbolos que se transmiten por segundo, y debe ser coherente en todos los elementos de la red para que puedan comunicarse.
- Dirección destino a FFFF (comando *ATDL FFFF*): de este modo el módulo XBee puede recibir mensajes de cualquier otro módulo XBee.

También consulta la dirección de 64 bits del nodo (comandos *ATSH* y *ATSL* para los bits más y menos significativos respectivamente), puesto que es usada como identificador en algunos mensajes, tal y como se ha descrito en el *Apartado 5.1.2* referente al formato de las comunicaciones.

La biblioteca XBee dispone de las siguientes subclases y funciones:

Subclases de la biblioteca XBee

- **Direccion64bits**: esta clase contiene dos atributos públicos uint32_t *DH* y uint32_t *DL,* ambos dos enteros sin signo de 32 bits destinados a almacenar los bits más significativos y menos significativos respectivamente de una dirección de 64 bits. Estos objetos son utilizados por la biblioteca XBee y los programas de control para facilitar el manejo de las direcciones de 64 bits. Además, esta clase también implementa una función denominada *get64()* que devuelve un entero sin signo de 64 bits formado por la concatenación de los valores DH y DL.

- **DatosFrame**: esta clase se utiliza para gestionar la información recibida en un mensaje de forma sencilla evitando tener que acceder a determinadas posiciones de bits cada vez que sea necesario extraer uno de los datos recibidos. Recoge todos los valores útiles que se reciben con el mensaje.

  Está compuesta por 10 atributos públicos:

  - *uint8_t APIid*: ID de la API que se utiliza para identificar el tipo de mensaje y por ende los campos que lo componen y su tamaño.
  - *uint8_t comando[2]*: si se trata de un comando AT, en esta variable se almacenan los dos caracteres correspondientes al nombre del comando.

- *uint8_t status*: en esta variable se indica si un comando AT se ha procesado correctamente.

- *uint16_t origen16*: si el mensaje procede de una dirección de 16 bits, dicha dirección se almacena en este campo.

- *Direccion64bits origen64*: referencia a un objeto *Direccion64bits* con la dirección del nodo origen si el mensaje procede de una dirección de 64 bits.

- *uint8_t rssi*: representa la intensidad de la señal existente entre el emisor del mensaje y el nodo. Puede utilizarse como indicador de distancia.

- *uint8_t fid*: ID del frame, puede utilizarse para realizar una cuenta de los frames recibidos y solicitar un reenvío si se pierden, o como ID para un posible ACK.

- *uint8_t datosTam:* indica el total de bytes de datos recibidos.

- *uint8_t* datos*: un array de caracteres que contiene los datos que el emisor quiere hacer llegar al receptor.

- *uint8_t op*: guarda el byte de opciones del mensaje. Este byte por defecto se utiliza para indicar si el emisor espera un ACK o no.

Constantes y variables de la biblioteca XBee

- **Constantes:**

- *TAM_LISTAS_STATUS 48*: tamaño de los arrays de ACKs y respuestas a comandos AT.

- *MAX_FRAME_TAM 115*: tamaño máximo en bytes que se reservan para un frame.

- *DESTINO_COORDINADOR_RED '1'*: cabecera 0x31.

- *CONFIRMA_NODO_REMOTO '2'*: cabecera 0x32.

- *DESTINO_NODO_RED '3'*: cabecera 0x33.

- *ACK '4'*: cabecera 0x34.

- *QUIERO_DATOS_COORDINADOR '5'*: cabecera 0x35.

- *DATOS_COORDINADOR '6'*: cabecera 0x36.

- *SELECCIONADO_COMO_ENRUTADOR '7'*: cabecera 0x37.

- *HELLO_COORDINADOR '8'*: cabecera 0x38.

- *DATOS '9'*: cabecera 0x39.

- **Variables públicas:**

- *uint16_t ID16*: variable que almacena la dirección de 16 bits del nodo.

- *Direccion64bits ID64*: almacena la dirección de 64 bits del nodo.

- *Direccion64bits direccionBroadcast*: dirección broadcast cuyo valor es 0x000000000000FFFF.

- **Variables privadas:**

- *SerialManager puerto_serial:* referencia a un objeto del tipo SerialManager.

- *uint8_t rcmdAT :* puntero del array de respuestas de comandos AT locales.

- *uint8_t* rcmdAT_id[TAM_LISTAS_STATUS]:* array con los ID de los comandos locales recibidos. El ID está representado por el nombre del comando.

- *uint8_t rcmdAT_status[TAM_LISTAS_STATUS]:* array complementario a *rmcdAT_id* que contiene el estado de la respuesta del comando local: 1 OK y 0 si no se ha podido procesar el comando.

- *uint8_t ack:* puntero del array de ACK.

- *uint8_t ack_id[TAM_LISTAS_STATUS]:* array con el ID de un ACK del que se espera respuesta. EL ID es un entero sin signo.

- *uint8_t ack_status[TAM_LISTAS_STATUS]:* array complementario a *ack_id* que contiene el estado de la respuesta del ACK enviado: 1 si positivo y 0 si negativo (NACK).

Funciones de la biblioteca XBee:

- **Públicas:**

- *void XBee():* es el constructor de la clase, tiene como función inicializar algunos parámetros como, por ejemplo, la dirección de broadcast.

- *void setPuertoSerial(SerialManager sm):* recibe como parámetro un objeto del tipo SerialManager que es usado para leer y escribir en el puerto serie.

- *bool configXbee(uint8_t* panid, uint32_t baudios):* autoconfigura el módulo XBee mediante comandos AT. Recibe como parámetros el ID de la red y la tasa de baudios de las comunicaciones. Devuelve *true* si el proceso concluye con éxito.

- *bool serial_broadcast(uint8_t* msg, uint8_t msgTam, uint8_t op): e*nvía un broadcast. Recibe como parámetros el mensaje a enviar, el tamaño del mensaje y el valor del byte de opciones. Devuelve *true* si el envío se realiza correctamente.

- *bool serial_ATlocal(uint8_t* comando, uint8_t* datos, uint8_t datosTam):* confecciona y envía un comando AT al propio módulo XBee. Recibe como parámetros el comando (dos caracteres), un valor si se trata de un comando de configuración y el tamaño del valor (si es 0 significa que no hay parámetro). Devuelve *true* si el envío se realiza con éxito.

- *bool serial_tx64(Direccion64bits dir64, uint8_t* msg, uint8_t msgTam, uint8_t fid, uint8_t op):* confecciona y envía un mensaje a una dirección de 64 bits. Recibe como parámetros la dirección destino, el mensaje a enviar, el tamaño del mensaje, el ID del frame y el byte de opciones. Devuelve *true* si el envío se realiza con éxito.

- *bool serial_tx16(uint16_t dir16, uint8_t* msg, uint8_t msgTam, uint8_t fid, uint8_t op):* igual que la función *serial_tx64(...)* pero usando direcciones de 16 bits.

- *bool serial_ack64(Direccion64bits dir64, uint8_t ackid, uint8_t status):* envía un ACK a una dirección de 64 bits. Recibe como parámetros la dirección destino, el ID del ACK y el status del ACK (1/0). Devuelve *true* si el envío se realiza con éxito.

- *bool serial_ack16(uint16_t dir16, uint8_t ackid, uint8_t status):* igual que la función *serial_ack64(...)* pero usando direcciones de 16 bits.

- *bool serial_rx(uint32_t timeout, DatosFrame& drx):* permanece a la espera de un mensaje el tiempo determinado por *timeout*. Cuando recibe un mensaje almacena los caracteres a la vez que des-escapa los caracteres especiales. Recibe como parámetro el tiempo de espera o *timeout* y una referencia a un objeto del tipo *DatosFrame* en el que deposita los datos extraídos del mensaje. Devuelve *true* si se ha recibido un mensaje.

- *uint8_t status_rcmdAT(uint8_t* comando, bool retirar):* comprueba si se ha recibido la respuesta a un comando AT. Recibe como parámetros el ID del comando y un booleano indicando si una vez consultado el resultado se debe retirar la entrada. Devuelve *1* si el resultado del comando es OK, *0* en caso contrario y *-1* si no se ha recibido nada todavía.

- *uint8_t status_ack(uint8_t ackid, bool retirar):* comprueba si se ha recibido un ACK. Recibe como parámetros el ID del ACK y un booleano indicando si una vez consultado el resultado se debe retirar la entrada. Devuelve *1* si el resultado del comando es OK, *0* en caso contrario y *-1* si no se ha recibido nada todavía.

- *void upStatus_rcmdAT(uint8_t* comando, uint8_t status):* actualiza el estado de un comando. Recibe como parámetros el ID del comando y el nuevo estado.

- *void upStatus_ack(uint8_t ackid, uint8_t status):* actualiza el estado de un ACK. Recibe como parámetros el ID del ACK y el nuevo estado.

- **Privadas:**

- *bool serial_tx(uint8_t* frame, uint8_t frameTam):* transmite un mensaje confeccionado por una de las funciones *serial_tx16, serial_tx64* o *serial_ATlocal* byte a byte a través del puerto serie a la vez que escapa los caracteres especiales.

- *DatosFrame extraerDatosFrame(uint8_t* frame, uint8_t frameTam):* cuando se recibe un mensaje, se llama a esta clase para extraer todos los datos relevantes del mismo. Recibe como parámetros un array de bytes que se corresponde con el mensaje recibido y su tamaño. Devuelve un objeto del tipo *DatosFrame* con los valores extraídos.

- *uint32_t millis()*: esta función sólo está presente en la biblioteca XBee que utiliza la Raspberry Pi y tiene como objetivo sustituir a la función nativa de Arduino *millis( )* que devuelve una marca de tiempo en milisegundos.

A continuación, se describen las características de la biblioteca SerialManager.

Biblioteca SerialManager

El manejo del puerto serie es importante y necesario en este proyecto, ya que los módulos XBee utilizan este puerto para leer los datos que reciben y para depositar aquellos a enviar. Las bibliotecas SerialManager para Arduino y Raspberry Pi comparten las mismas funciones para que puedan ser utilizadas indistintamente por la biblioteca XBee. Por tanto, la diferencia radica en cómo estas funciones son implementadas, ya que mientras que para Arduino el manejo del puerto serie es trivial siendo únicamente necesario invocar a las funciones *read(), write()* e *isAvailable()* de *Serial*, para Raspberry Pi implica comunicarse con el sistema operativo (realizar la apertura del puerto serie y configurar cómo se de deben llevar a cabo las comunicaciones).

El objeto SerialManager es inicializado en cada una de las plataformas y pasado como argumento a la función *SetPuertoSerial(...)* de la biblioteca XBee:

- Para Arduino la forma de inicializar un objeto SerialManager consiste en iniciar el puerto serie normalmente con *Serial.begin(BAUDIOS)* y pasar este objeto como referencia al constructor de SerialManager: *bool setSerial(Stream &serial)*. Devuelve *true* si todo ha ido bien.

  El puerto serie se guarda como un puntero a un objeto de tipo Stream: *Stream* _Serial*.

- Para Raspberry Pi el constructor queda así: bool *setSerial(uint32_t BAUDIOS)* se comporta como el de Arduino pero la inicialización y configuración completa del puerto serie se lleva a cabo en la propia función. Devuelve *true* si todo ha ido bien.

  El puerto serie se guarda como un puntero a un fichero: *int32_t _Serial*.

El parámetro BAUDIOS indica la tasa de baudios a utilizar por el puerto, que debe ser el mismo que el de los módulos XBee.

El resto de funciones que implementa esta biblioteca son las siguientes:

- *uint8_t disponible( )*: indica si hay algún dato en el puerto serie esperando a ser leído. Devuelve el número de caracteres que esperen en el buffer de entrada.
- *uint8_t leer()*: lee un caracter del puerto serie y lo devuelve.
- *uint8_t flush( )*: vacía el buffer de entrada del puerto serie.
- *void escribir(uint8_t val)*: escribe el carácter pasado como parámetro en el puerto serie.
- *void cerrar( )*: cierra el puerto serie.

5.2.2 Controlador de los nodos y Sense (Noduino)

El controlador de los nodos gestiona la búsqueda del controlador cuando se une a la red, el envío de datos y el procesamiento de los mensajes recibidos.

Para guardar la configuración del coordinador utiliza las siguientes variables:

- *uint64_t enrutador*: cuando un nodo se convierte en enrutador suma uno a esta variable.
- *bool rutado*: si un nodo no tiene acceso al coordinador de la red y otro nodo asume dicha función (enrutador) este parámetro es *true*.
- *Direccion64bits direccionCoord*: dirección del coordinador / enrutador.
- *uint8_t rssiCoord*: RSSI entre el nodo y el coordinador / enrutador.
- *bool enrutDirecto*: si el nodo está siendo enrutado por un nodo vecino sin acceso directo al coordinador este valor es *true*.
- *bool coordDirecto*: si el coordinador del nodo es el coordinador de la red vale *true*.

También guarda los tiempos de espera o escucha, la tasa de baudios por defecto de la red y el ID de la red.

Funciones del controlador de los nodos:

- *void setup( )*: esta función debe ser implementada obligatoriamente en cualquier programa para Arduino. En ella se inicializan variables, el puerto serie, el objeto *XBee* y se llama a la función de configuración *configXbee( )* de la biblioteca *XBee*.

- *void loop( )*: también de obligada implementación, esta función se repite en bucle y es la función principal de los programas para Arduino. Se mantiene a la escucha de mensajes provenientes de otros nodos o del coordinador durante un tiempo determinado. A continuación, si tiene acceso a un coordinador transmite los datos de sus sensores y, si no, inicia la búsqueda de un coordinador. Tras realizar estas acciones pasa a un modo de bajo consumo durante un periodo de tiempo establecido.

- *bool escuchar(uint32_t timeout):* se mantiene a la espera de varios mensajes el tiempo indicado en el parámetro *timeout*. Cada vez que recibe uno llama a la función *procesarFrame( )* para que lo procese. Devuelve *true* si ha recibido algún mensaje.

- *uint8_t buscarVecinos(uint32_t timeout)*: manda broadcasts preguntando a los nodos cercanos por los datos de su conexión al coordinador y guarda las respuestas en una cola FIFO. Devuelve el total de vecinos que han respondido.

- *bool getVecino(DatosFrame& drx)*: extrae el primer registro almacenado en la cola de vecinos. Devuelve *true* si la cola no estaba vacía.

- *bool actualizarCoordinador( )*: de entre todos los vecinos que hayan respondido a la petición de datos se escoge uno de ellos y se le indica que ha sido escogido como coordinador. Si responde pasa a ser el coordinador del nodo. Devuelve *true* si se ha actualizado el coordinador.

- *bool comprobarCoordinador( )*: es llamado en cada ciclo de la función *loop( )* si el nodo está activo y siendo enrutado. Cada 5 minutos envía un mensaje al coordinador para comprobar si sigue activo o si ha recibido una respuesta. Si en tres intentos no la recibe pasa a inactivo.

- *bool txCoordinadorRed(uint8_t* datos, uint8_t datosTam, uint8_t fid, uint8_t op)*: envía un mensaje directamente al coordinador de la red. Si el acceso al coordinador de la red no es directo empaqueta los datos a enviar con la cabecera correspondiente y la dirección origen, en caso contrario se envía normalmente. Devuelve *true* si el envío se ha realizado con éxito.

- *bool ackCoordinadorRed(uint8_t ackid, uint8_t status)*: semejante a *txCoordinadorRed* pero enviando un ACK.

- *bool txDatos()*: Transmite los datos de los sensores al coordinador, devolviendo *true* si el envío se ha realizado con éxito.

- *void procesarFrame(DatosFrame& drx)*: recibe un objeto del tipo *DatosFrame* de un mensaje entrante y, en función del encabezado de los datos, lleva a cabo una acción determinada que puede incluir una respuesta al nodo emisor. Devuelve *true* si se ha podido procesar el mensaje.

- *bool checkTimeout(uint32_t &timer, uint32_t ms, bool reset)*: función de apoyo para comprobar si un contador ha llegado a 0 y, en el caso de que sea cíclico, reiniciarlo. Recibe como parámetros el tiempo acumulado, el tiempo total a esperar y un booleano indicando si se debe reiniciar al llegar a 0. Devuelve *true* si el temporizador ha finalizado.

Bibliotecas complementarias:

Como se ha mencionado en la introducción, se han utilizado dos bibliotecas para complementar algunas funcionalidades. Estas bibliotecas, a diferencia de las anteriores, no han sido desarrolladas, sino que se han obtenido de los repositorios públicos de Arduino:

- SnoozeLib: esta biblioteca permite poner a Arduino en un modo de bajo consumo durante un período determinado de tiempo. Lo que hace es poner la CPU en el modo SLEEP_MODE_PWR_DOWN. En este modo, el microcontrolador consume tan solo 0,36 mA ya que deshabilita casi toda la circuitería. Para despertar a la CPU, la biblioteca emplea un timer denominado Watchdog Timer, ya que es el único que permanece habilitado en este modo al estar implementado en un oscilador separado del resto de timers de Arduino [12].

- QueueList: proporciona la implementación de una cola FIFO enlazada y los métodos comunes a este tipo de estructuras: *flush( ), pop( )* y *push( )*.

Módulo Sense

El módulo Sense es una extensión del controlador para los nodos Arduino en el que se indican los pines correspondientes a los sensores, si están habilitados o no y sus identificadores. El objetivo de esta separación es que sea más sencillo alternar entre diferentes conjuntos de sensores, ya que se pretende que este sistema de monitorización del entorno sea lo suficientemente versátil como para adaptarse a cualquier situación. En la *Figura 5.21*, a modo de ejemplo, se muestra una representación de la asociación de diferentes conjuntos de sensores.

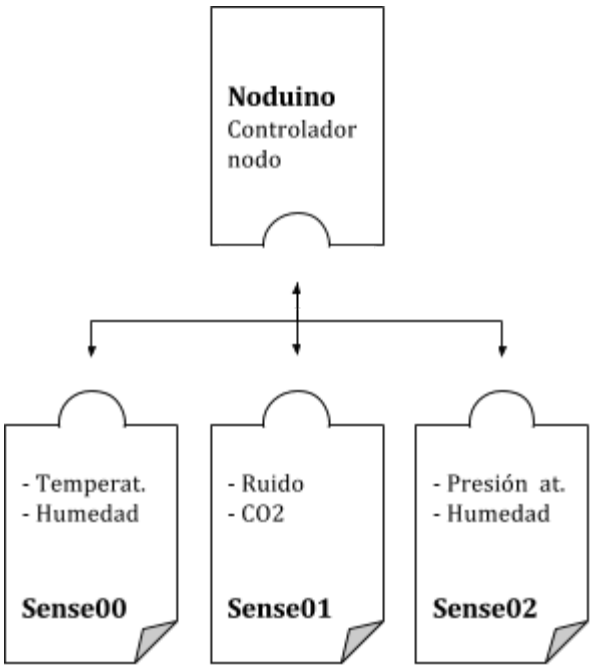


Figura 5.21: Asociación de módulos Sense al controlador del nodo

En estos módulos se define una función denominada *leerSensores( )*, a la que el controlador de los nodos Arduino llama cuando va a realizar un envío de datos al coordinador, que devuelve una lectura de los datos en forma de array de caracteres estructurado, así como el tamaño total de dicho array. Este array contiene la información leída de cada uno de los sensores junto a su identificador.

5.2.3 Controlador del coordinador (PiCo)

El controlador del coordinador controla el acceso de nuevos nodos a la red, comprobando que hayan sido registrados y no se encuentren activos y hace de intermediario entre los nodos y el servidor.

Cuando recibe un mensaje de datos de sensores los almacena en un fichero denominado *fichero de datos*. Los datos de este fichero son reenviados al servidor cada minuto y se mantienen hasta que se confirma que se encuentran correctamente almacenados en la base de datos. Si no se puede establecer conexión con el servidor, los datos se acumulan hasta que alcanzan los 100.000 registros, momento en el cual se comienzan a reescribir las líneas desde el principio del fichero. Con esto se pretende evitar que el tamaño del fichero exceda el máximo administrable por el sistema o incluso consuma toda la memoria del mismo provocando un error y dejando al coordinador fuera de servicio.

También se mantiene un fichero denominado *log* en el que se registran los eventos que tienen lugar en el sistema. La mecánica de funcionamiento de este fichero es similar a la del *fichero de datos*.

Variables del controlador del coordinador:

El controlador del coordinador almacena la configuración de diversos parámetros:

- La tasa de baudios a la que debe configurarse el módulo XBee.
- Identificador de la red (PANID).
- Máximo número de líneas a escribir en los ficheros de log y datos antes de sobreescribirlos.
- Rutas a los ficheros de log y datos.
- Tiempos de espera para envío de datos.

Funciones del controlador del coordinador:

- *int main( ):* ejecuta un bucle infinito en el que comprueba si se ha llegado algún mensaje para mandarlo a procesar o si ha concluido el timeout que indica que deben enviarse los datos recopilados al servidor. Devuelve 0 si el programa terminó normalmente y -1 en caso de error.

- *void setup( ): i*nicia el puerto serie y el objeto XBee, a continuación llama a la función de autoconfiguración de la biblioteca XBee y, por último, crea los ficheros de datos y log.

- *char* timestamp( ):* obtiene una marca de tiempo del sistema que incluye día, mes, año y hora para usarla en los logs de eventos y datos. Devuelve una cadena de caracteres.

- *void log(char* nuevalinea, bool timestamp_on, ofstream& fich):* inserta una nueva línea en el fichero de datos o en log y la muestra a través de la consola.

- *bool almacenarDatos( ):* inserta los datos del fichero de datos en la base de datos del servidor. Devuelve *true* si la inserción ha concluido con éxito.

- *bool procesarFrame(DatosFrame& drx):* recibe un objeto del tipo *DatosFrame* de un mensaje entrante y, en función del encabezado de los datos, lleva a cabo una acción determinada que puede incluir una respuesta al nodo emisor. Devuelve *true* si se ha podido procesar el mensaje.

- *bool comprobarNodo(Direccion64bits nodo, bool validar, Direccion64bits nodoCoord)*: comprueba un nodo en la base de datos. Recibe como parámetros el identificador del nodo a comprobar, un booleano indicando si se debe forzar una actualización del estado en la red y validarlo en la base de datos, y el identificador del nodo enrutador (si lo está). Devuelve *true* si el nodo existe en la base de datos y no ha sido validado o si ha sido validado correctamente, y *false* si cualquiera de estas dos condiciones no se cumple.

- *uint8_t* getRuta(Direccion64bits nodo):* obtiene la ruta hacia un nodo remoto y la devuelve en forma de array de caracteres.

5.2.4 Frontend

El *frontend* web utiliza PHP y el controlador de MongoDB para PHP para interactuar con la base de datos, y javascript para algunas de las funciones del sitio web.

Puesto que el objetivo del proyecto es el desarrollo de la red de sensores, no se ha hecho hincapié en proporcionar un frontend con un gran surtido de funcionalidades y opciones. Se ha procurado que en la versión actual ofrezca las funcionalidades básicas que permiten administrar este sistema.

Así pues el *frontend* web presenta cinco secciones, además de la página principal:

- Página principal: un menú con botones de acceso a cada una de las secciones.

- Visualización de datos: en ella se muestran los datos recopilados por los nodos. En esta sección es posible ver los datos transmitidos por cada nodo, filtrarlos bajo diferentes criterios y eliminar entradas.

- Administración de nodos: se muestra el listado de nodos y coordinadores del sistema. Permite insertar nuevos nodos y coordinadores, deshabilitarlos, eliminarlos y modificar los datos de los que ya están registrados. Además, permite visualizar en un mapa la situación de todos ellos y filtrarlos bajo diferentes criterios.

- Administración de sensores: se muestra un listado de los sensores registrados en la base de datos. Permite insertar nuevos sensores, modificarlos, deshabilitarlos y eliminarlos.

- Administración de usuarios: permite visualizar los usuarios registrados en el sistema, así como añadir nuevos usuarios y deshabilitar o eliminar a los que ya existen.

- Administración de alertas: permite visualizar, crear, modificar, deshabilitar y eliminar alertas.

Gestor de Alertas

Dentro del *frontend* también se incluye un programa denominado *Gestor de Alertas* que se encarga de comprobar las alertas configuradas en la base de datos y los datos relacionados con estas para proceder a desencadenar la acción indicada si se cumplen las condiciones definidas (en el estado actual de la red de monitorización esta acción implica enviar un email).

La configuración de las alertas se guarda en la base de datos como una cadena de texto con el formato:

ID_NODO | ID_SENSOR | VALOR_MAX | VALOR_MIN | VALOR_C | SEPARADOR

Dónde:

- ID_NODO: es el identificador del nodo en el que se encuentra el sensor a monitorizar.
- ID_SENSOR: es el identificador del sensor monitorizado.
- VALOR_MAX: por encima de este valor se desencadenará la alerta.
- VALOR_MIN: por debajo de este valor se desencadenará la alerta.
- VALOR_C: un valor específico para el que se desencadenará la alerta.
- SEPARADOR: separa dos configuraciones de sensores diferentes.

Una alerta puede incluir la monitorización de un nodo y varios sensores. Si no se desea indicar uno de los valores VALOR_MAX, VALOR_MIN ó VALOR_C se utilizará el carácter '*' en lugar de un valor concreto para indicar que dicho campo no debe tenerse en cuenta.

El software de gestión de alertas no dispone de interfaz gráfica. La gestión de las alertas se lleva a cabo en la correspondiente sección del *frontend*.

5.3 Diseño de la interfaz

En este proyecto el diseño de la interfaz se refiere al *frontend* web para administrar el sistema. Puesto que se pretende centrar el desarrollo en la red de sensores, éste es sencillo y cuenta con las cinco secciones ya mencionadas: visualización de datos, gestión de nodos, gestión de sensores, gestión de alertas y gestión de usuarios, además de la página principal, los formularios y un mapa para visualizar la situación de los nodos desplegados.

El diseño del *frontend* se ha llevado a cabo en dos partes. En primer lugar, como se describe en la sección 5.3.1, se ha definido su estructura de forma esquemática, adelantando algunas de las funcionalidades previstas. A continuación, se ha procedido a la maquetación del mismo y se ha definido su funcionalidad al completo, tal y como se describe en la sección 5.3.2.

5.3.1 Diseño esquemático de la interfaz

La página principal, cuya representación puede verse en la *Figura 5.22*, pretende tener la misma función que el menú principal de una aplicación, permitiendo seleccionar entre las cinco secciones del *frontend*:

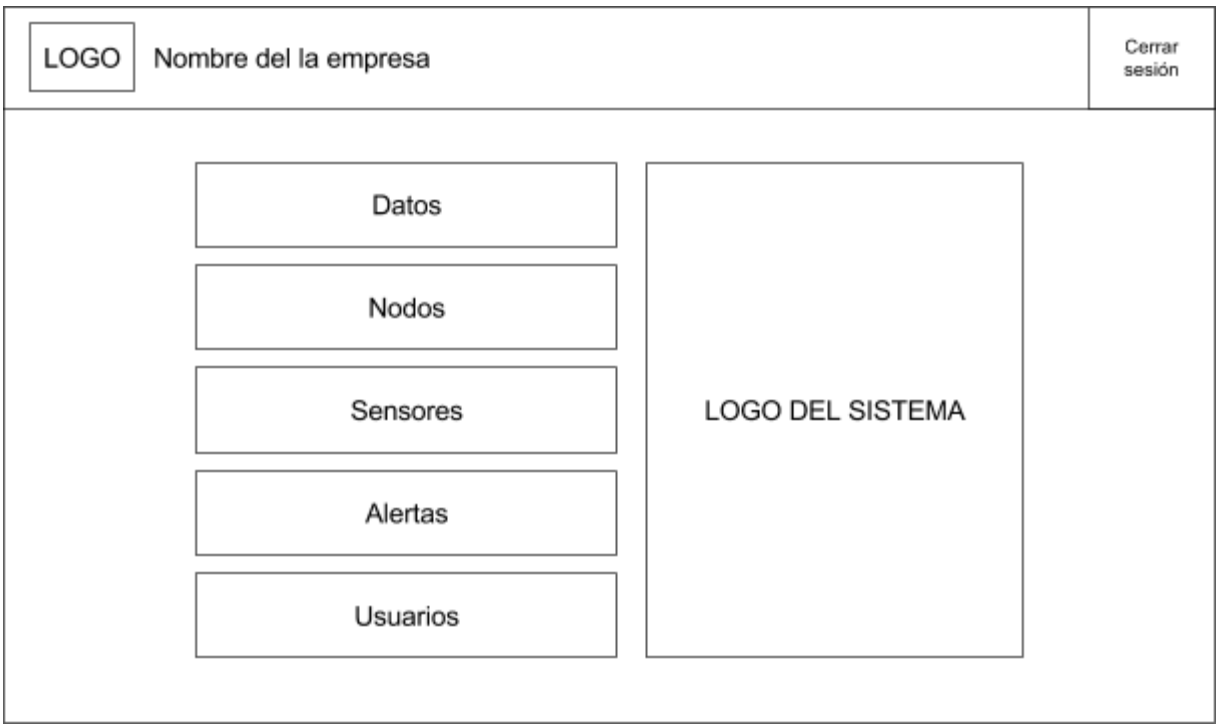


Figura 5.22: Boceto de la pantalla principal del *frontend*

De la *Figura 5.22* cabe destacar los siguientes elementos:

- El logo, situado en la esquina superior izquierda, está pensado para mostrar el logotipo de la institución o empresa que emplee el sistema.
- Por otra parte, el logo del sistema, ubicado en el lateral derecho del menú principal, tiene como objetivo mostrar el logotipo comercial del sistema.

Para simplificar la gestión del sistema, todas las secciones se han planteado con una disposición similar:

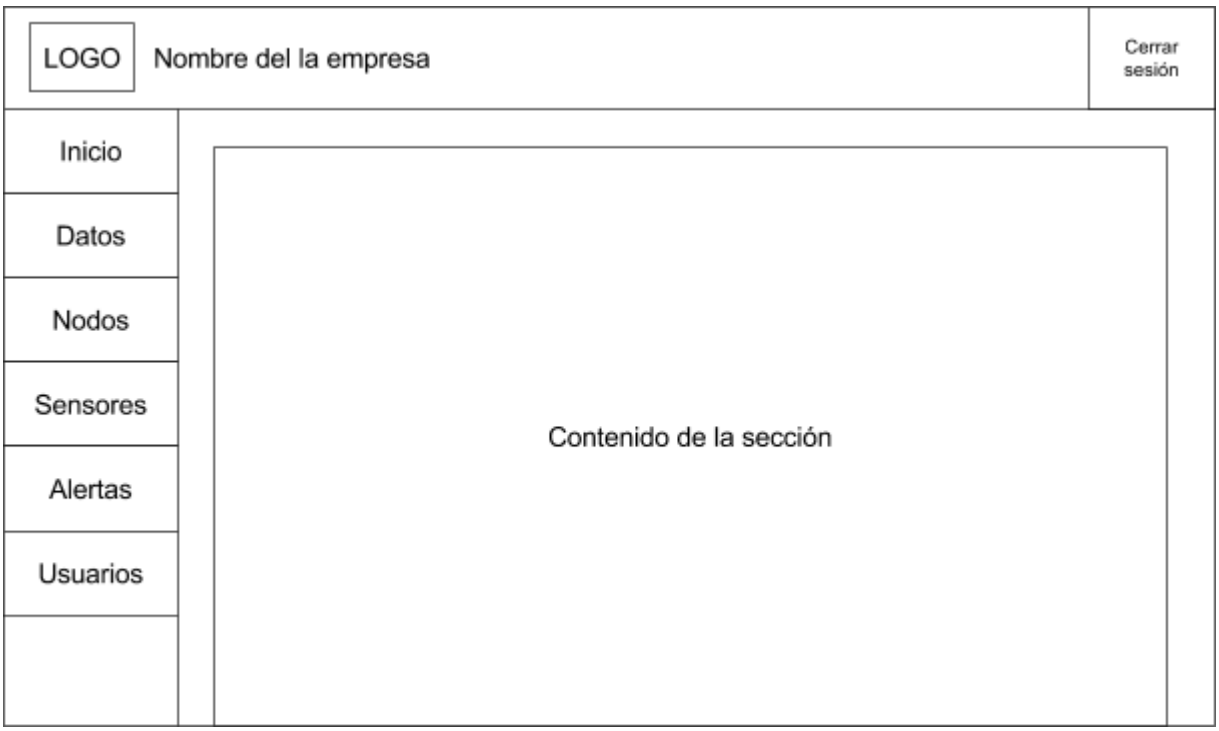


Figura 5.23: Boceto de una sección genérica recogidos por los sensores

En la *Figura 5.23* se pueden observar las siguientes características:

- En el banner superior se ubican el nombre del sistema, un posible logotipo de la empresa y el botón de cierre de sesión.

- El botón de cierre de sesión ("Cerrar sesión") se encuentra en la esquina superior derecha, separado del menú principal (en el lateral izquierdo), para evitar ser pulsado accidentalmente.

- En el lateral izquierdo se muestra una barra que hace la función de menú de navegación de la página. Este menú muestra botones a las diferentes secciones de la web, incluyendo un botón extra que permite regresar a la pantalla de inicio.

- El resto del espacio, que supone la mayor parte del área disponible, queda dedicado al contenido de las secciones.

Con este esquema genérico para todas las secciones, se pretende crear un entorno que resulte familiar y que permita que el usuario pueda ubicarse fácilmente en cualquiera de ellas.

Además, puesto que en todas las secciones se presenta una lista de elementos, los cuerpos de las mismas también compartirán una estructura similar. Dicha estructura constará de un título con el nombre de la sección, junto a un botón para añadir nuevos elementos, y una lista con entradas para cada uno de los elementos de la correspondiente sección que se encuentren almacenados en la base de datos. Cada una de estas entradas dispondrá a su vez de botones para editarlas, bloquearlas/deshabilitarlas o eliminarlas. La disposición de estos elementos puede verse en la *Figura 5.24*:

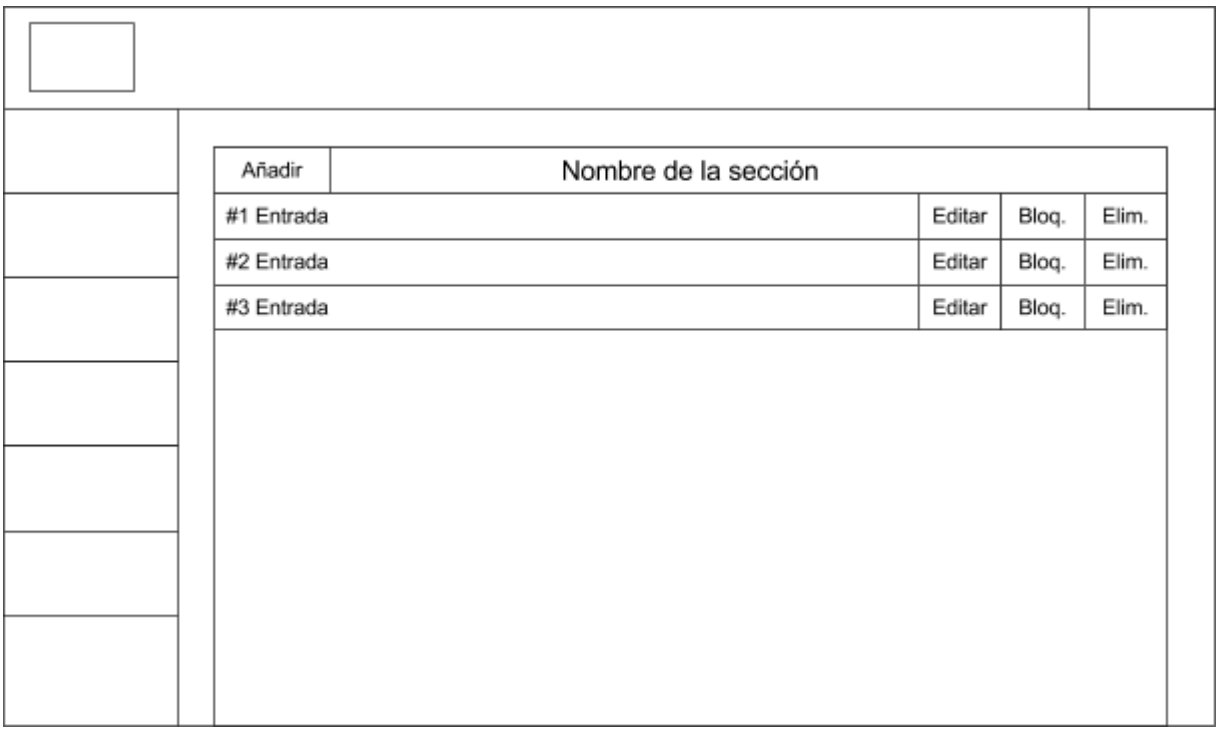


Figura 5.24: Boceto de una lista genérica del frontend

Estas entradas podrán mostrar sólo texto o incluir elementos dinámicos embebidos como enlaces y mapas. Además, en función de las características del elemento o de los permisos del usuario, algunas entradas de la lista no mostrarán todos los botones mencionados.

Respecto al conjunto de la sección, todos los elementos de la misma permanecerán estáticos a excepción de la lista, a través de la cual se podrá realizar un desplazamiento vertical cuando el número de entradas sobrepase el tamaño de la página.

También se han diseñado sendos formularios para cada uno de los cuatro tipos de datos que es posible administrar: nodos, sensores, alertas y usuarios.

En primer lugar, en la *Figura 5.25*, se muestra la estructura del formulario para la creación y edición de nodos y coordinadores:

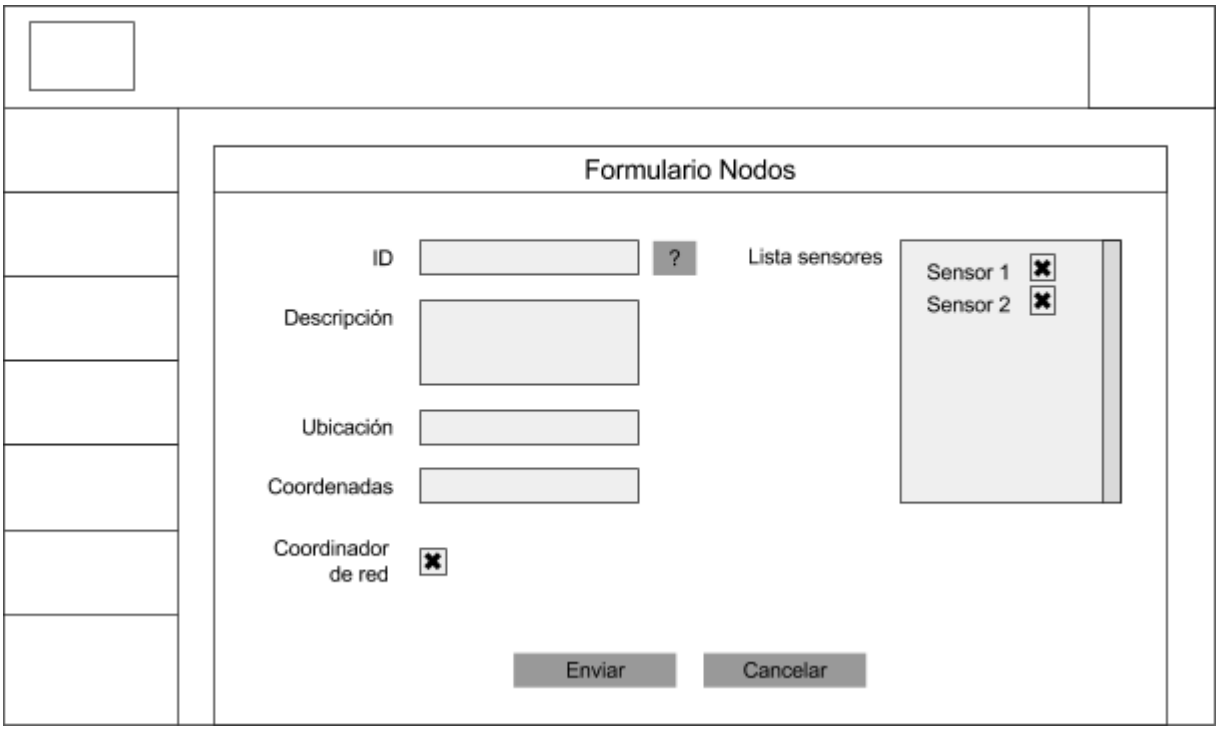


Figura 5.25: Boceto del formulario de nodos

Este formulario permite modificar en cualquier momento el identificador del nodo (ID), el cual se corresponde con la dirección de 64 bits usada por los módulos XBee para transmitir. El motivo de esto es que dicha dirección podría cambiar en caso de ser necesario sustituir al módulo debido a un fallo, avería o cambios en el diseño hardware de los nodos.

Por otra parte, junto al campo del identificador del nodo (ID) se incluye un botón (“?”) cuyo objetivo es mostrar, a modo de ayuda, una imagen indicando donde se ubica la dirección MAC, o de 64 bits, del módulo XBee.

Respecto a la lista de sensores, sólo deberá mostrarse si el elemento registrado no está marcado como coordinador, ya que estos no poseen sensores.

En la *Figura 5.26* se muestran los campos del formulario de los sensores:

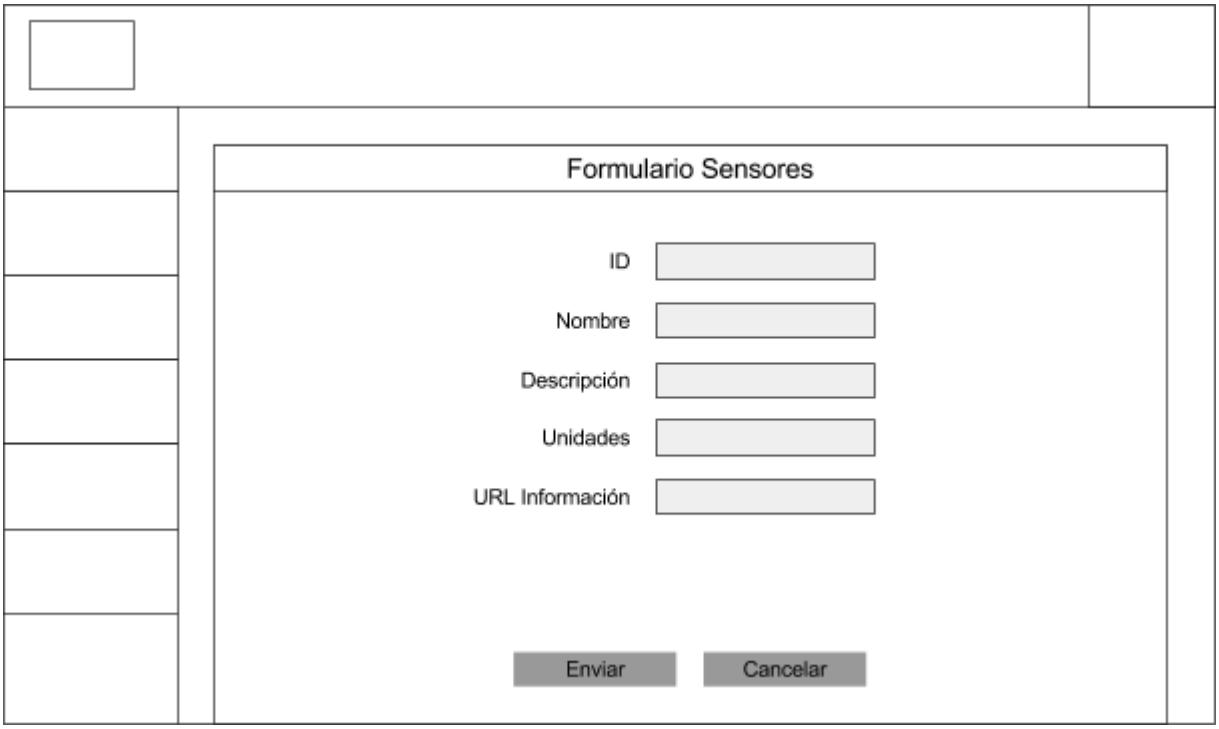


Figura 5.26: Boceto del formulario de sensores

Este formulario permite definir el identificador del sensor sólo la primera vez, en sucesivas modificaciones no se mostrará este campo. Dicho identificador debe además coincidir con el que se especifique en los módulos Sense.

Por otra parte, el campo "URL Información" pretende ser utilizado para incluir un enlace al datasheet del sensor, siempre que éste se encuentre disponible, o en su defecto a una página informativa sobre las características del sensor.

El formulario para crear y modificar alertas se ha diseñado tal y como puede verse en la *Figura 5.27*:

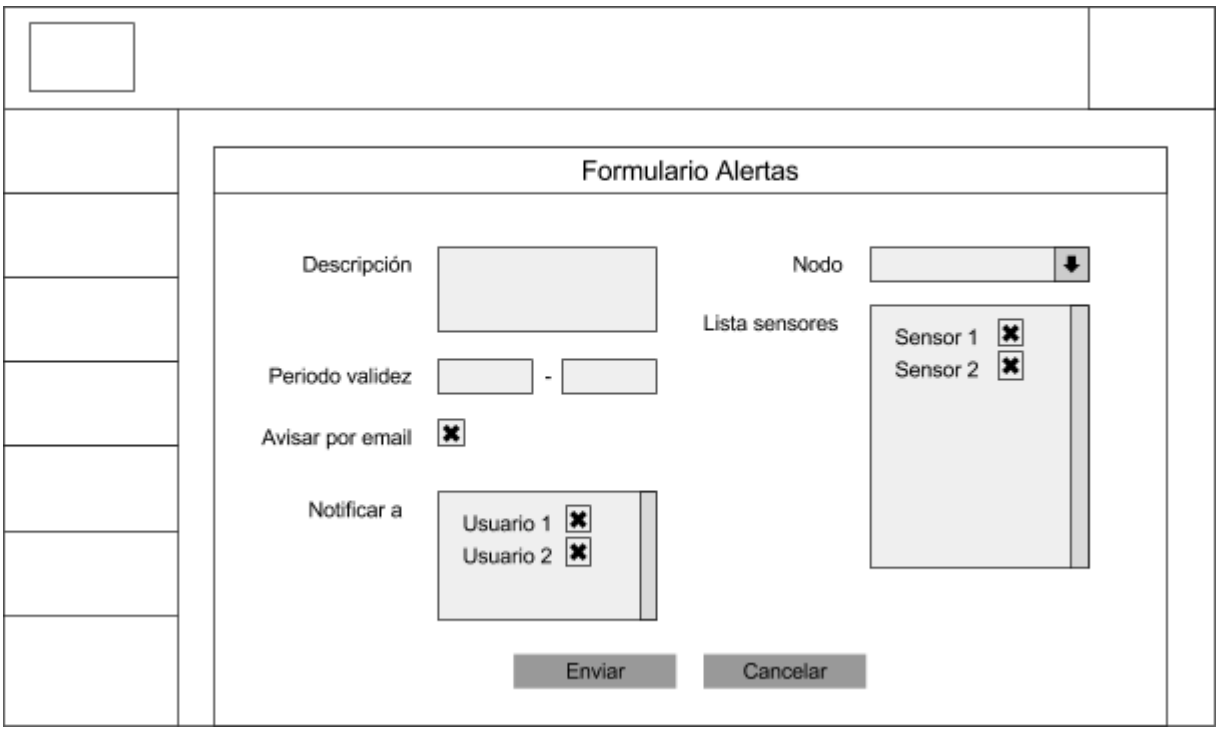


Figura 5.27: Boceto del formulario de alertas

La lista de sensores, que se muestra en el lateral derecho de este formulario, mostrará únicamente al conjunto de sensores instalados en el nodo seleccionado mediante el desplegable que se encuentra justo encima de la misma.

Del mismo modo, la lista de usuarios en el lado izquierdo del formulario, referenciada como "Notificar a", tan sólo mostrará a aquellos usuarios que no se encuentren dados de baja en el sistema.

Por último, en la *Figura 5.28* se presenta el formulario para la creación y modificación de los datos de los usuarios:

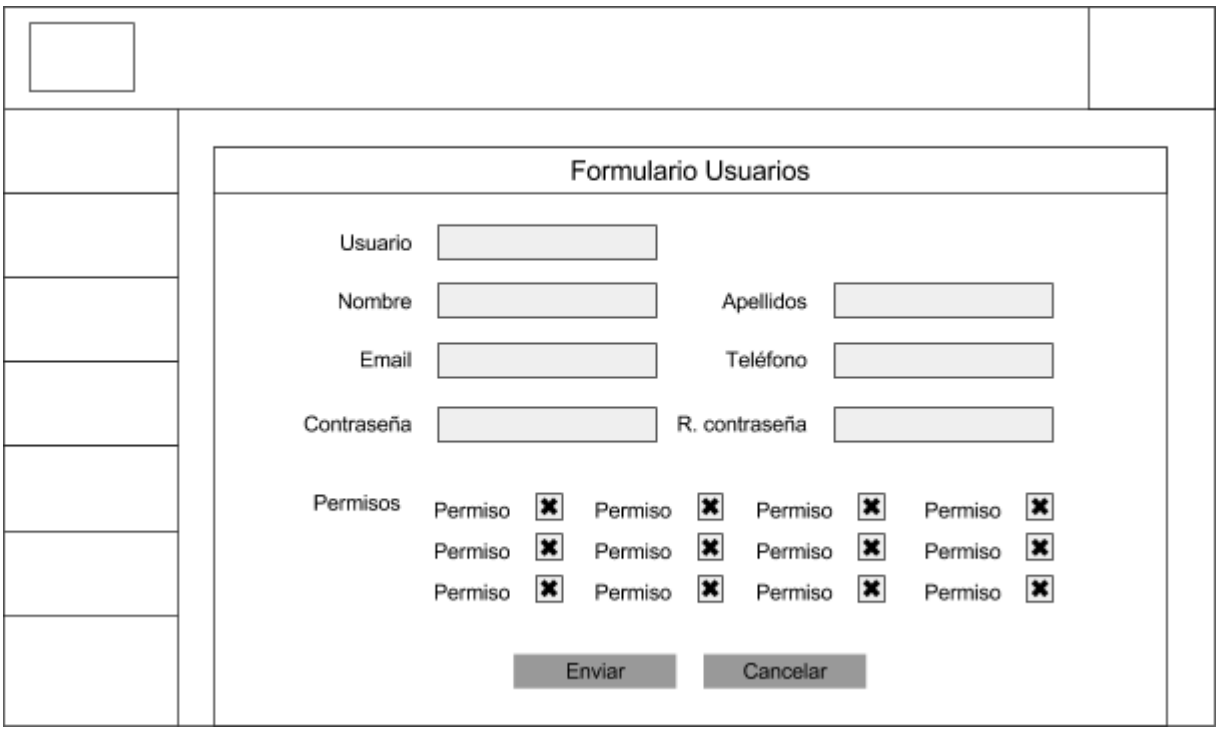


Figura 5.28: Boceto del formulario de usuarios

Al igual que el campo ID del formulario de sensores, el nombre de usuarios sólo podrá definirse la primera vez.

Respecto al listado de permisos que se muestran desglosados en el conjunto de casillas que aparecen en la parte inferior de este formulario, estos pretenden hacer referencia a las cinco secciones: datos, nodos, sensores, alertas y usuarios, y a cada uno de los tres tipos de acciones que se pueden llevar a cabo sobre ellas: creación, modificación y eliminación de elementos.

5.3.2 Maquetación de la interfaz

A continuación, se muestra el diseño del *frontend* definitivo a partir de los bocetos e indicaciones especificados. Además, para cada sección, se detallan sus características y funcionalidades más relevantes.

En primer lugar, para realizar el inicio de sesión, se ha diseñado una página que se muestra previamente a la página principal si el usuario todavía no ha iniciado sesión. Como se puede ver en la *Figura 5.29,* esta página muestra un sencillo formulario para introducir el nombre de usuario y la contraseña.

Figura 5.29: Pantalla de inicio de sesión del *frontend*

Si el usuario se encuentra dado de baja o alguno de los dos campos no es correcto, se muestra un mensaje de error. En la *Figura 5.30* se muestra una captura de dichos errores:

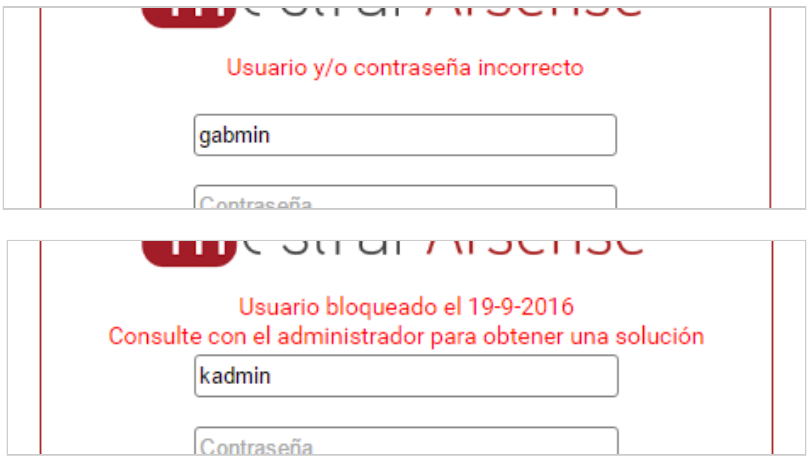


Figura 5.30: Errores en el inicio de sesión

Una vez el usuario ha iniciado sesión correctamente, es llevado a la página principal. En esta página se muestra un menú con acceso a las diferentes secciones.

En la cabecera, aparece el nombre de la empresa que está haciendo uso del sistema junto a un logo, si se quiere incluir, seguido de la palabra "Arsense" que es el nombre que recibe el sistema desarrollado en este proyecto.

En la *Figura 5.31* puede observarse una captura de la pantalla de inicio:

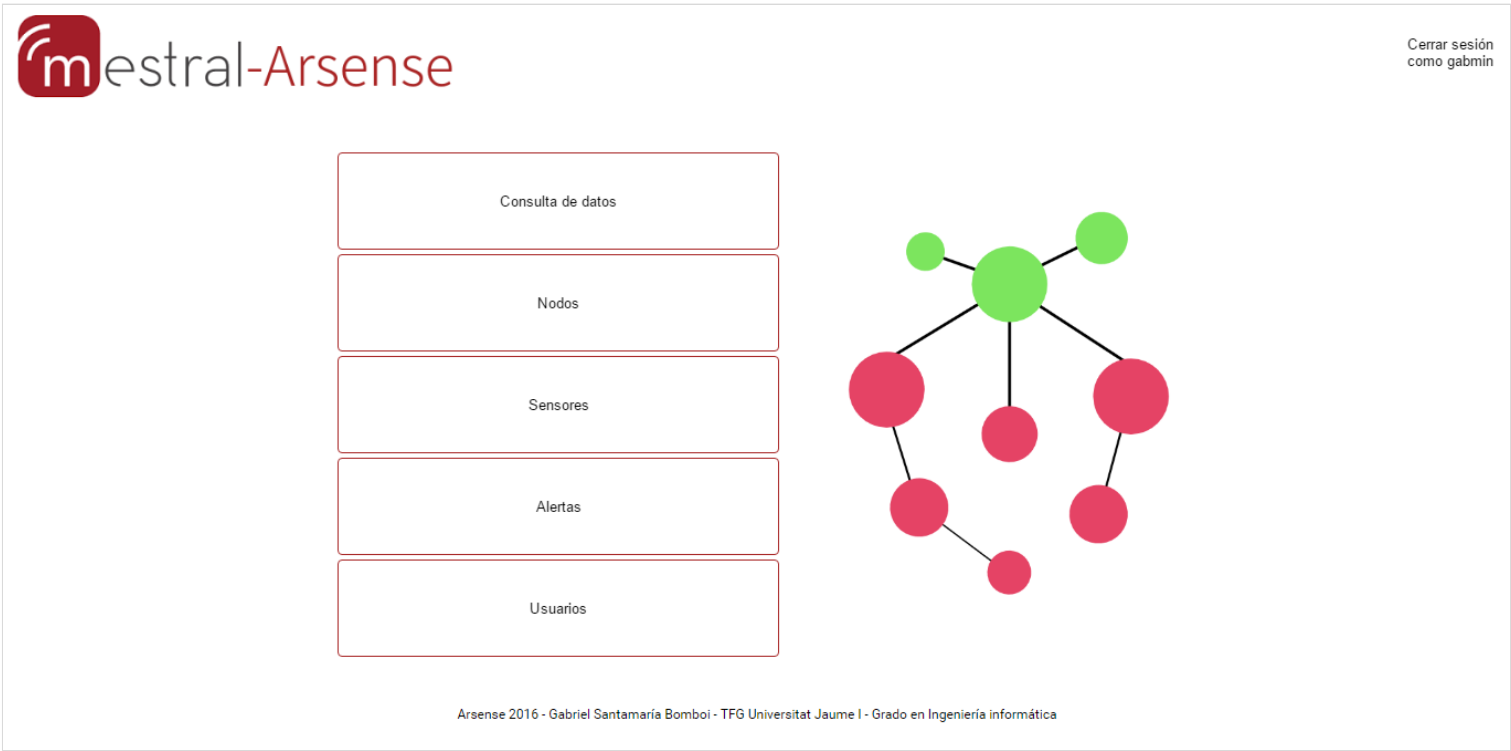


Figura 5.31: Pantalla de inicio del *frontend*

En lo que respecta al logo del sistema, este representa un conjunto de nodos cuya disposición y colores pretenden evocar al logotipo de Raspberry Pi, tal y como puede apreciarse en la comparativa de la *Figura 5.32*:

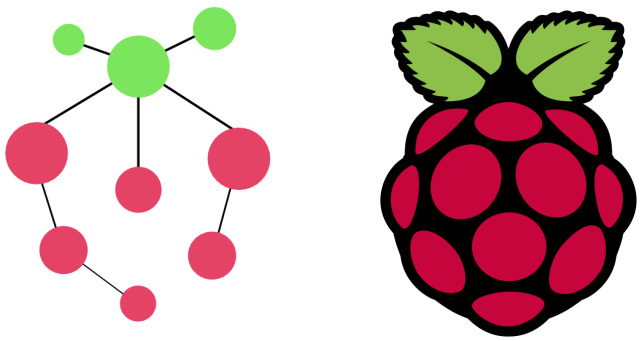


Figura 5.32: Logotipo del sistema

A continuación, se van a describir las cinco secciones del *frontend* junto a sus formularios:

- Sección de visualización de datos:

  Como se puede ver en la *Figura 5.33*, se muestran los datos recibidos en una tabla en la que cada entrada representa la información de un sensor para un determinado nodo. Se ha determinado que los datos sólo pueden eliminarse, por ejemplo para casos en los que se hayan realizado pruebas o se hayan recibido datos corruptos, pero no editar o crear ya que en un sistema de recogida de datos dichas acciones no tienen sentido.

  Además, en la parte superior se muestra un pequeño formulario que sirve para realizar un filtrado completo de los datos por cada uno de sus campos. Puede verse en detalle en la *Figura 5.34*.

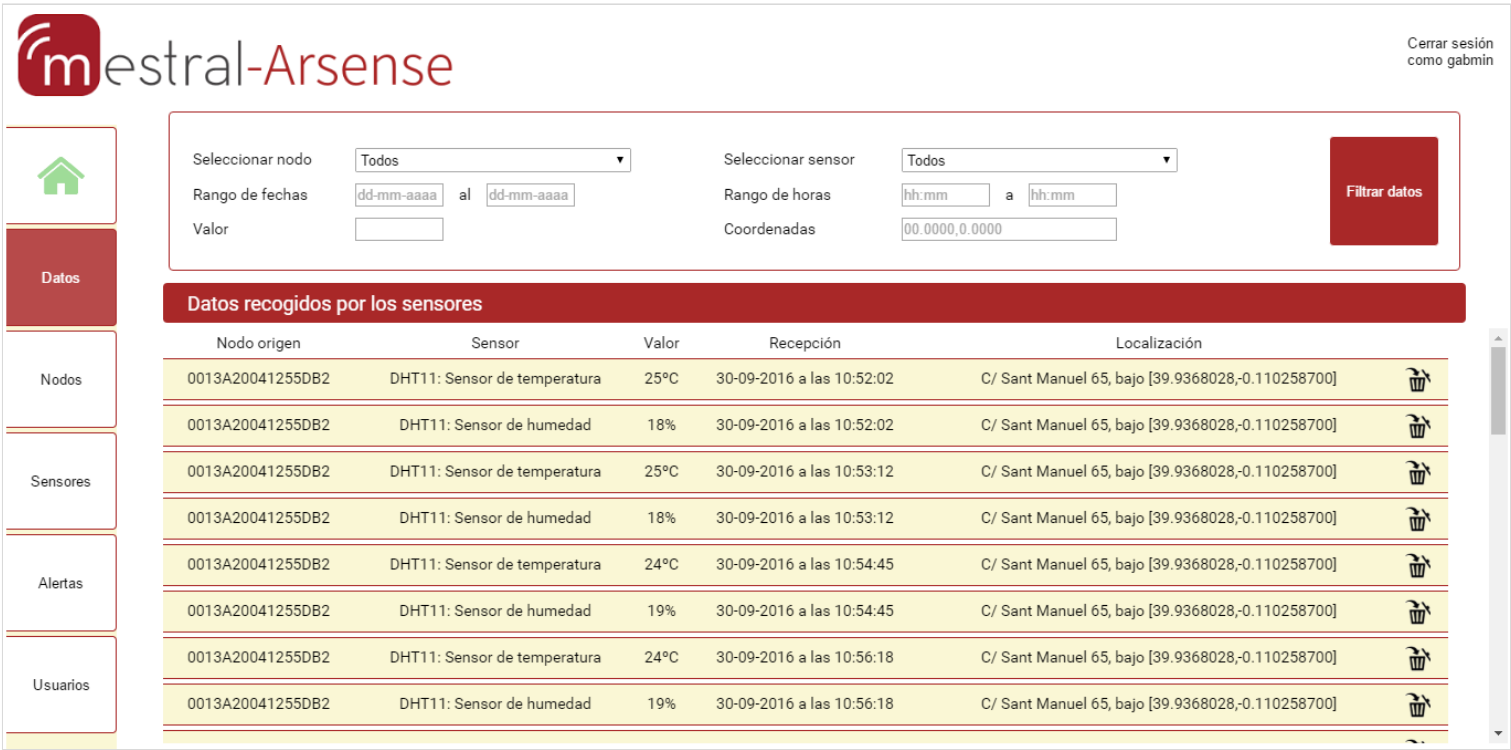


Figura 5.33: Sección de visualización de datos del *frontend*

Figura 5.34: Formulario de filtrado de datos

Existen una serie de elementos comunes a todas las secciones: los cuadros de diálogo y los descriptores de los botones.

Los primeros se utilizan para informar al usuario de un error en la introducción de datos o el procesamiento de una determinada acción, así como para confirmar la ejecución de acciones comprometidas, tales como eliminar o bloquear elementos. En la *Figura 5.35* se muestran dos ejemplos de las ventanas emergentes que aparecen al tratar de bloquear y eliminar un elemento.

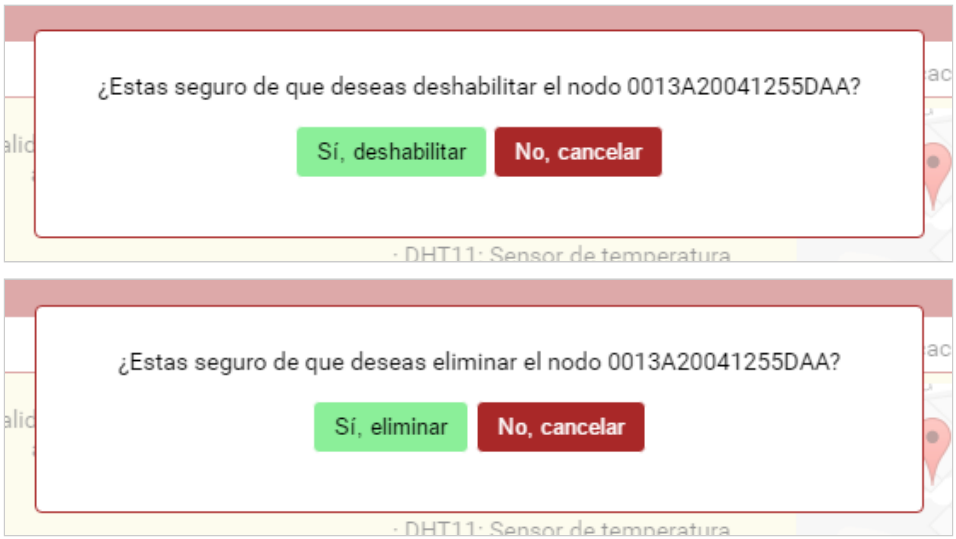


Figura 5.35: Ventanas emergentes al deshabilitar (superior) o eliminar (inferior) un elemento

Los descriptores son pequeños textos que aparecen cuando el usuario posa el cursor sobre un botón. Como puede verse en la *Figura 5.36* sirven para informar de la función del botón en cuestión.

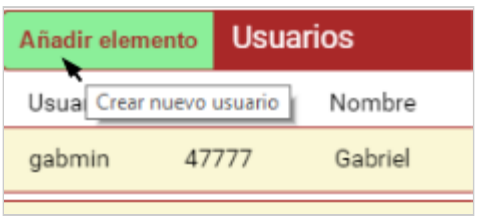


Figura 5.36: Descriptor de un botón

Además, algunos botones cambian su aspecto para resultar más informativos al usuario, bien sea iluminándose al posar el cursor sobre ellos o cambiando su aspecto, como es el caso del botón utilizado para habilitar/deshabilitar elementos que se muestra en la *Figura 5.37*:

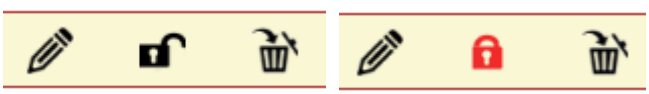


Figura 5.37: Cambio de aspecto en el botón de bloqueo

- Sección de administración de nodos:

  Como se muestra en la *Figura 5.38*, la sección de administración de nodos es similar a la de visualización de datos, aunque con algunas diferencias visuales: la ubicación de los nodos y coordinadores se muestra como un mapa interactivo de Google Maps, y estos últimos aparecen resaltados en azul para diferenciarse de los nodos.

  En la descripción aparece un breve texto explicativo sobre la función del nodo (si procede) además de los sensores de los que dispone.

  A diferencia de los datos, los nodos son elementos que pueden editarse, bloquearse (si, por ejemplo, están fuera de servicio) o eliminarse de la base de datos, por ello se muestra en cada entrada los tres botones que se corresponden con estas acciones. Además, se muestra un cuarto botón, cuyo icono es el de un marcador del mapa tachado, que aparece cuando un nodo está validado en la red, permitiendo invalidarlo de nuevo.

  El campo "Estado en la red" muestra la fecha y hora en la que el nodo se ha unido a la red (es decir, ha entrado en contacto con el coordinador de la red y éste lo ha validado contra la base de datos) y también si tiene acceso directo al coordinador ("conectado a través del coordinador <DIRECCIÓN>) o está siendo enrutado por otro nodo ("conectado a través de nodo enrutador <DIRECCIÓN>). En caso de no haber sido validado aparece el texto "No validado" y, si se trata de un coordinador, se muestra el texto "Coordinador de red".

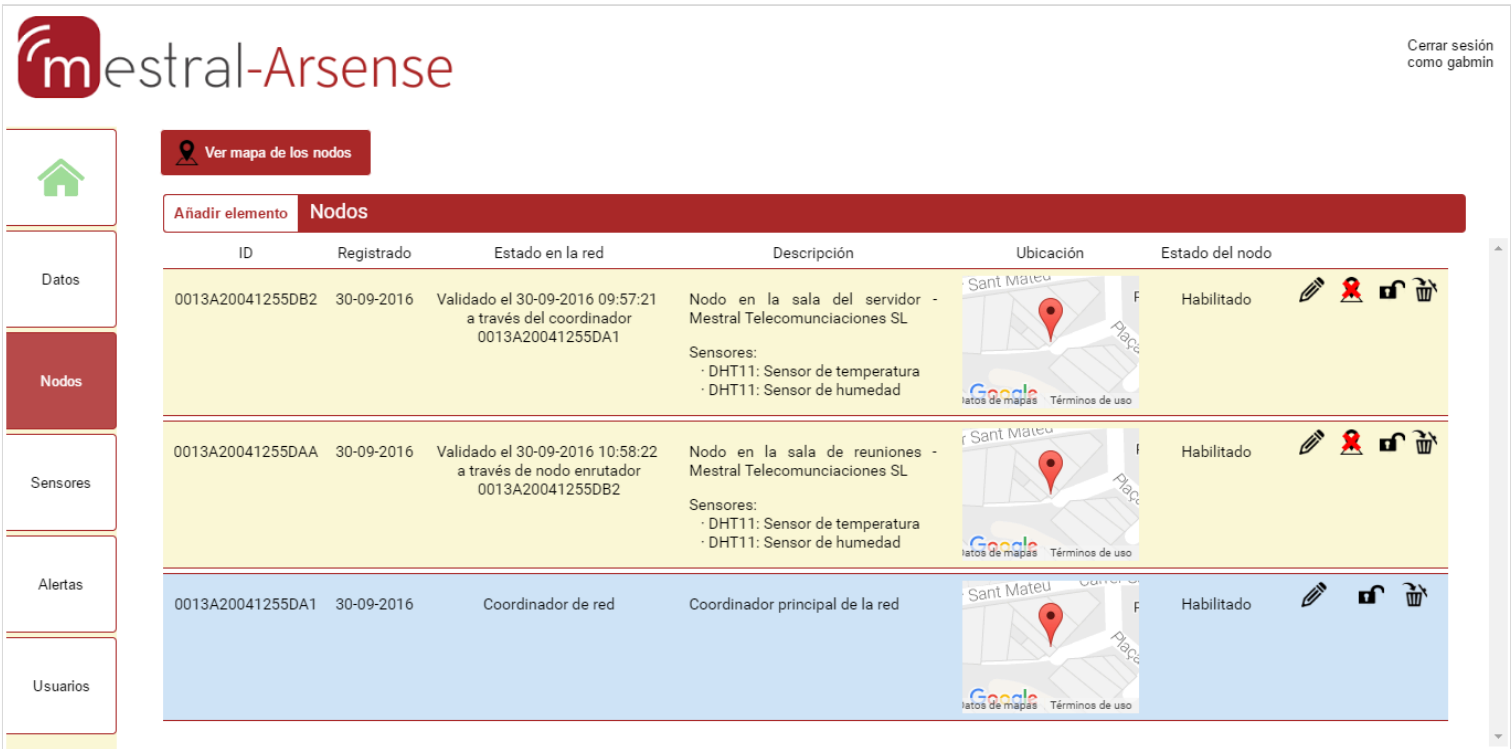


Figura 5.38: Sección de administración de nodos del *frontend*

En la parte superior de la tabla, se ubica un botón para acceder al mapa que se muestra en la *Figura 5.39*, el cual permite visualizar la situación de todos los nodos y coordinadores registrados en la red.

Los marcadores que determinan su posición aparecen diferenciados por colores en función de su estado o función (coordinador: azul, nodo validado: verde, nodo no validado: amarillo y deshabilitado: rojo). Además, los elementos mostrados pueden filtrarse en base a estos dos criterios utilizando el formulario que se muestra en el lateral izquierdo del mapa, y ubicarlos en él al clicar sobre su identificador en la lista que se muestra justo debajo de dicho formulario. También es posible clicar directamente sobre los marcadores del mapa para consultar la información de un nodo o coordinador, así como la última lectura realizada por los sensores de un nodo.

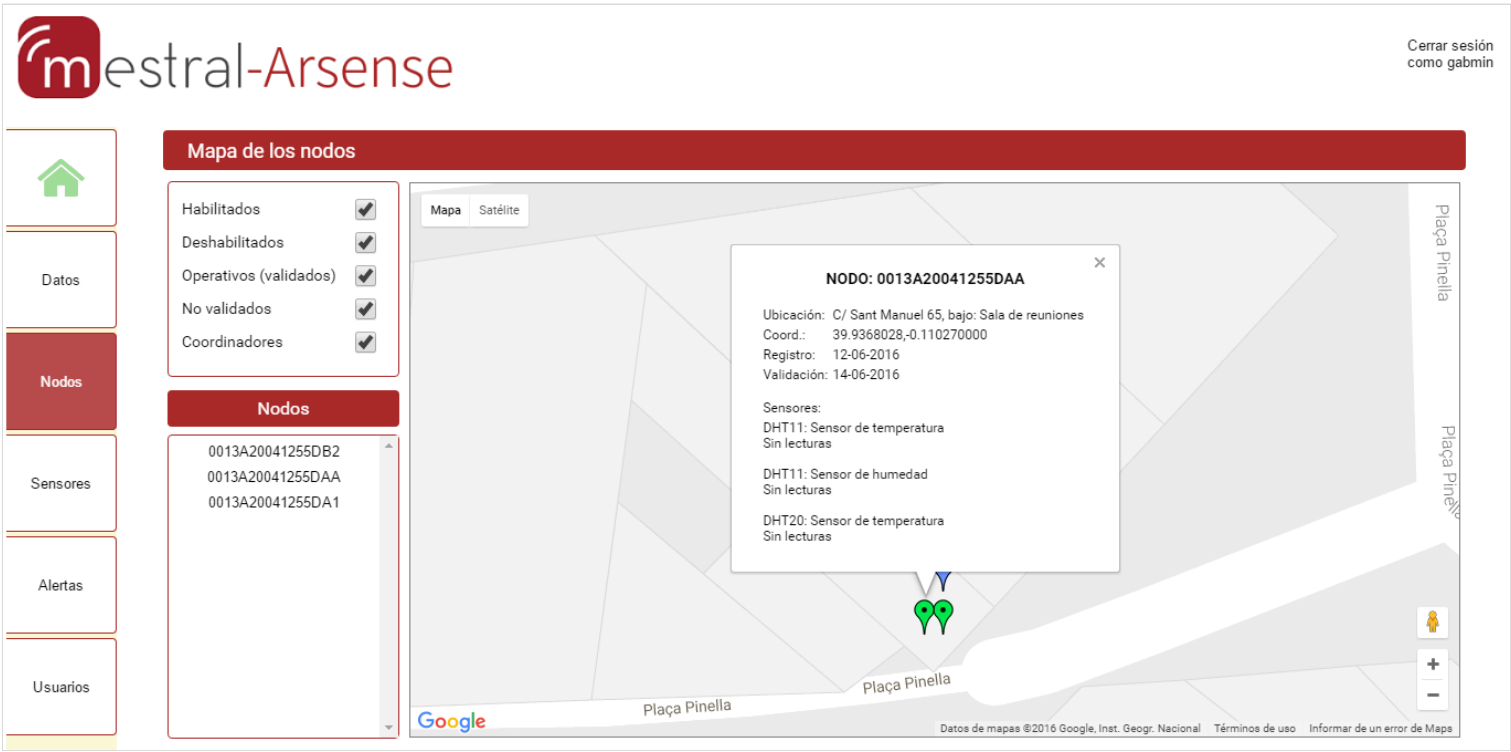


Figura 5.39: Mapa de los nodos desplegados

A continuación, en la *Figura 5.40* se muestra el formulario para registrar nuevos nodos en la base de datos o modificar la información de los ya existentes. Como se ha mencionado en el correspondiente boceto, este formulario permite modificar el ID del nodo ya que éste se corresponde con la dirección de 64 bits de su módulo XBee, la cual puede cambiar si se producen cambios en dicho hardware.

Por otra parte, en la lista de sensores aparecen únicamente aquellos que no se encuentran bloqueados. No obstante, si un nodo que tiene un modelo de sensor bloqueado en su configuración, éste aparecerá en rojo para indicar al usuario que ya no está disponible o no es válido. En la *Figura 5.41* se muestra cómo aparece uno de estos sensores bloqueados. Este comportamiento se extiende a todos los formularios y secciones que tratan con elementos susceptibles de ser bloqueados/deshabilitados.

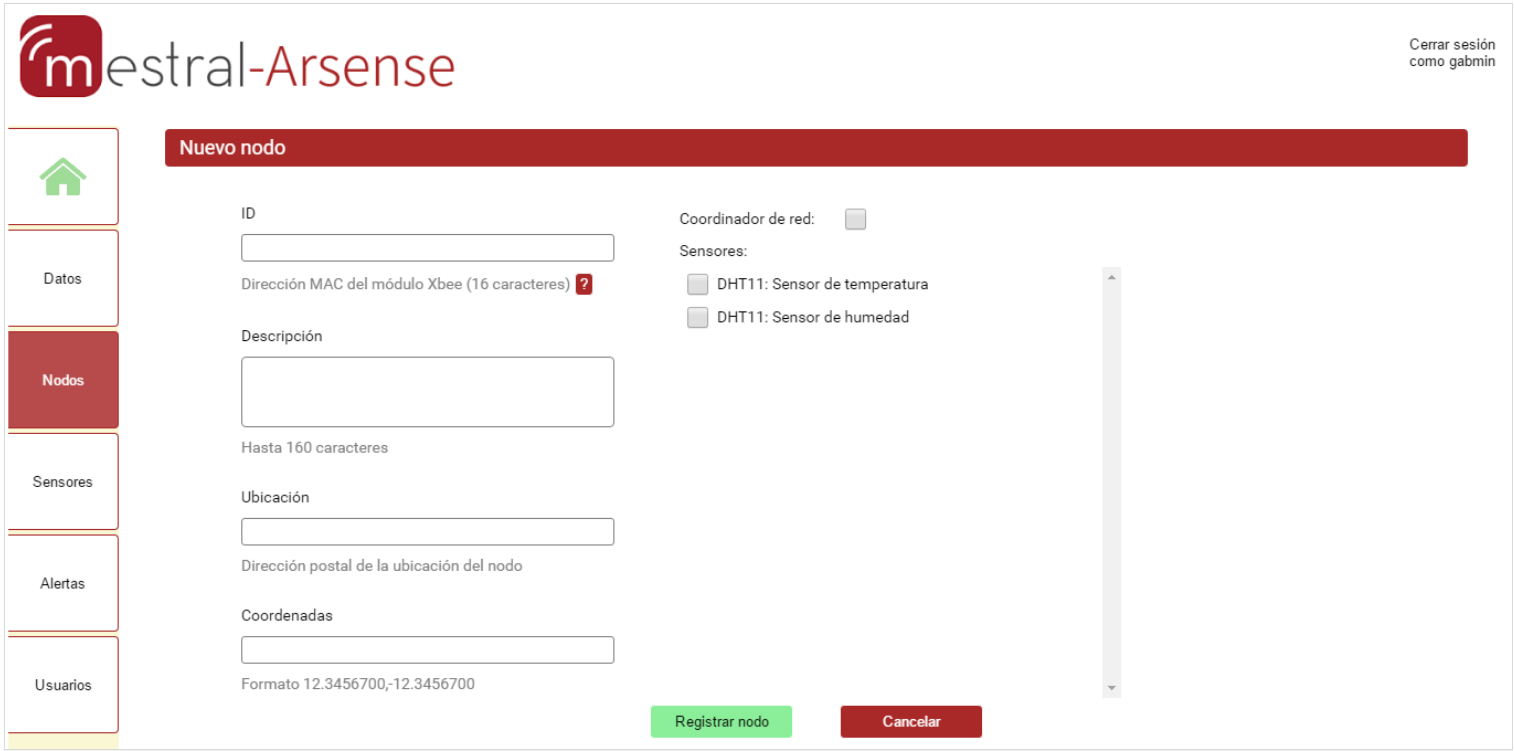


Figura 5.40: Formulario de registro/modificación de nodos

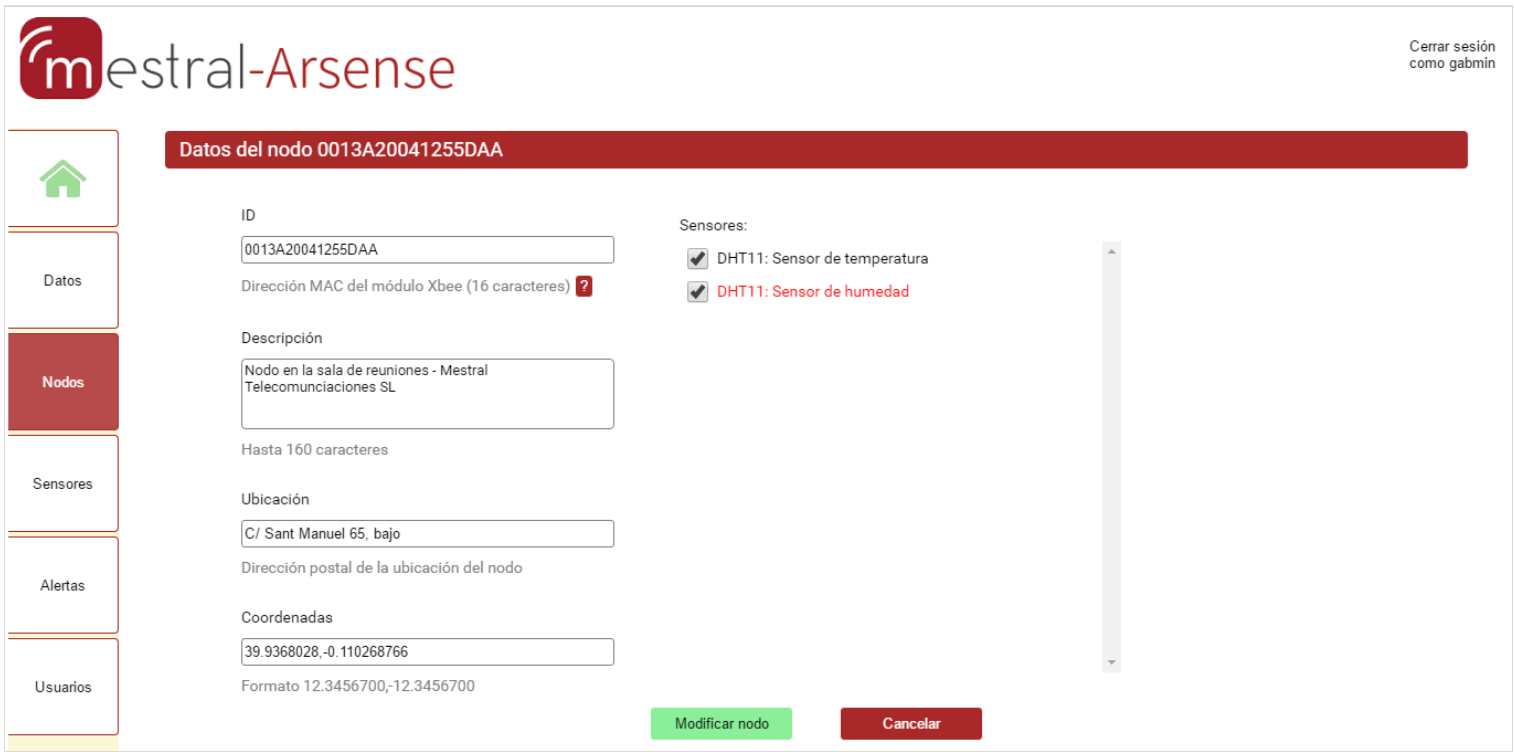


Figura 5.41: Ejemplo de un sensor bloqueado al modificar los datos de un nodo

Como se aprecia en la *Figura 5.41* cuando se edita un elemento la información vigente aparece en los campos correspondientes. Además, si se borra, persiste como una marca de agua en el fondo del campo.

Como se menciona en el boceto correspondiente a este formulario, junto al campo "ID" se muestra un botón de ayuda que, al ser clicado, despliega una ventana emergente mostrando en qué lugar de los módulos XBee aparece la dirección de 64 bits usada como identificador. Esta ventana se muestra en la *Figura: 5.42*:

Figura 5.42: Ventana emergente de ayuda para localizar el ID de un módulo XBee

Por último, el *checkbox* situado en la parte superior del listado de nodos, cuya etiqueta dice "Coordinador de red", es utilizado para indicar que se trata de un coordinador de red. Este atributo solamente puede ser establecido al crear el nodo y no puede cambiarse. Si se marca, el listado de sensores quedará oculto, pues un coordinador de red no está diseñado para ser utilizado para tareas de monitorización del entorno.

- Sección de administración de sensores:

Los sensores, como puede verse en la *Figura 5.43*, se muestran como un catálogo. Cada entrada dispone de botones para editar, deshabilitar/habilitar o eliminar una entrada.

El formulario para registrar y modificar sensores puede verse en la *Figura 5.44*. El valor indicado como ID debe coincidir con el utilizado en los módulos Sense para que el sistema pueda interpretar correctamente los datos registrados por el coordinador de red en la base de datos.

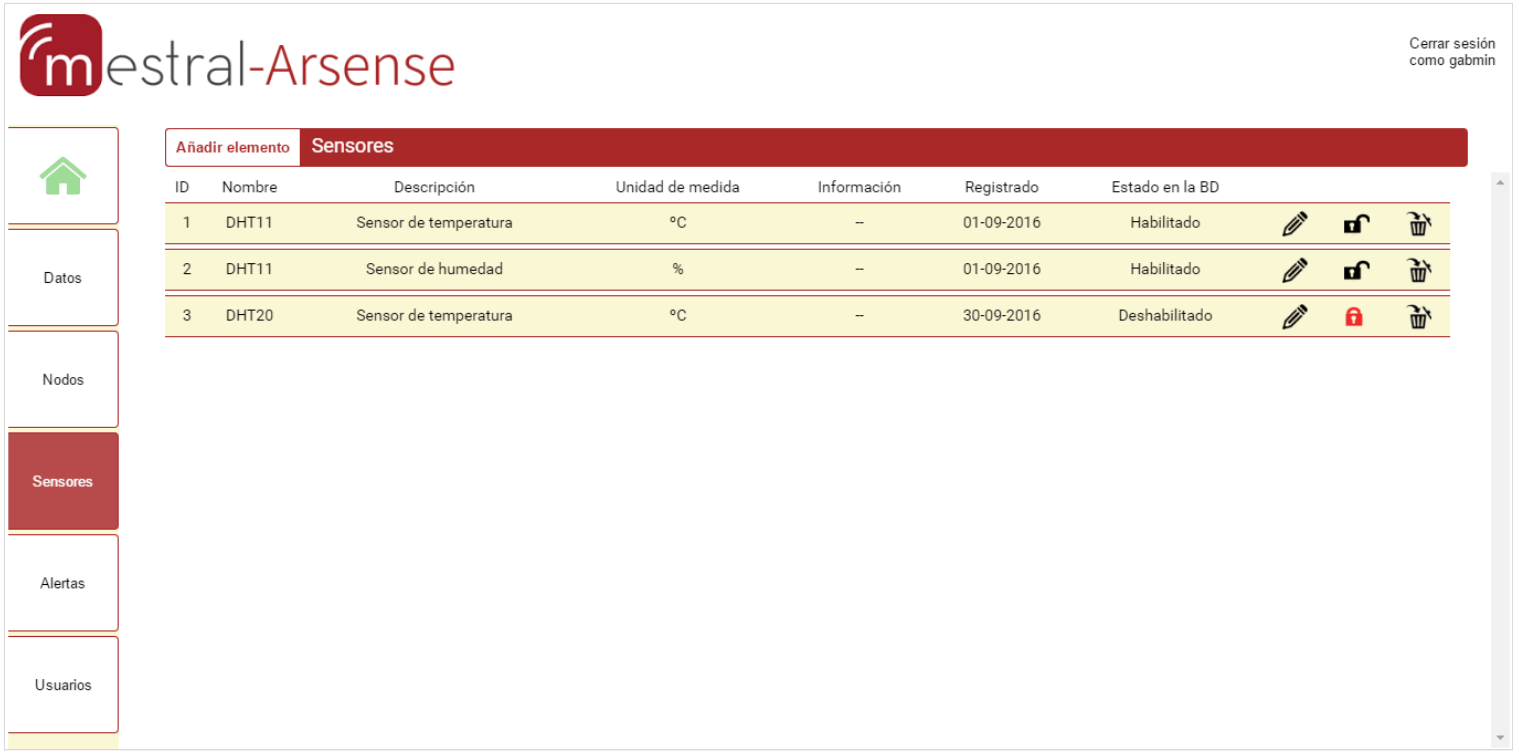


Figura 5.43: Sección de administración de sensores del *frontend*

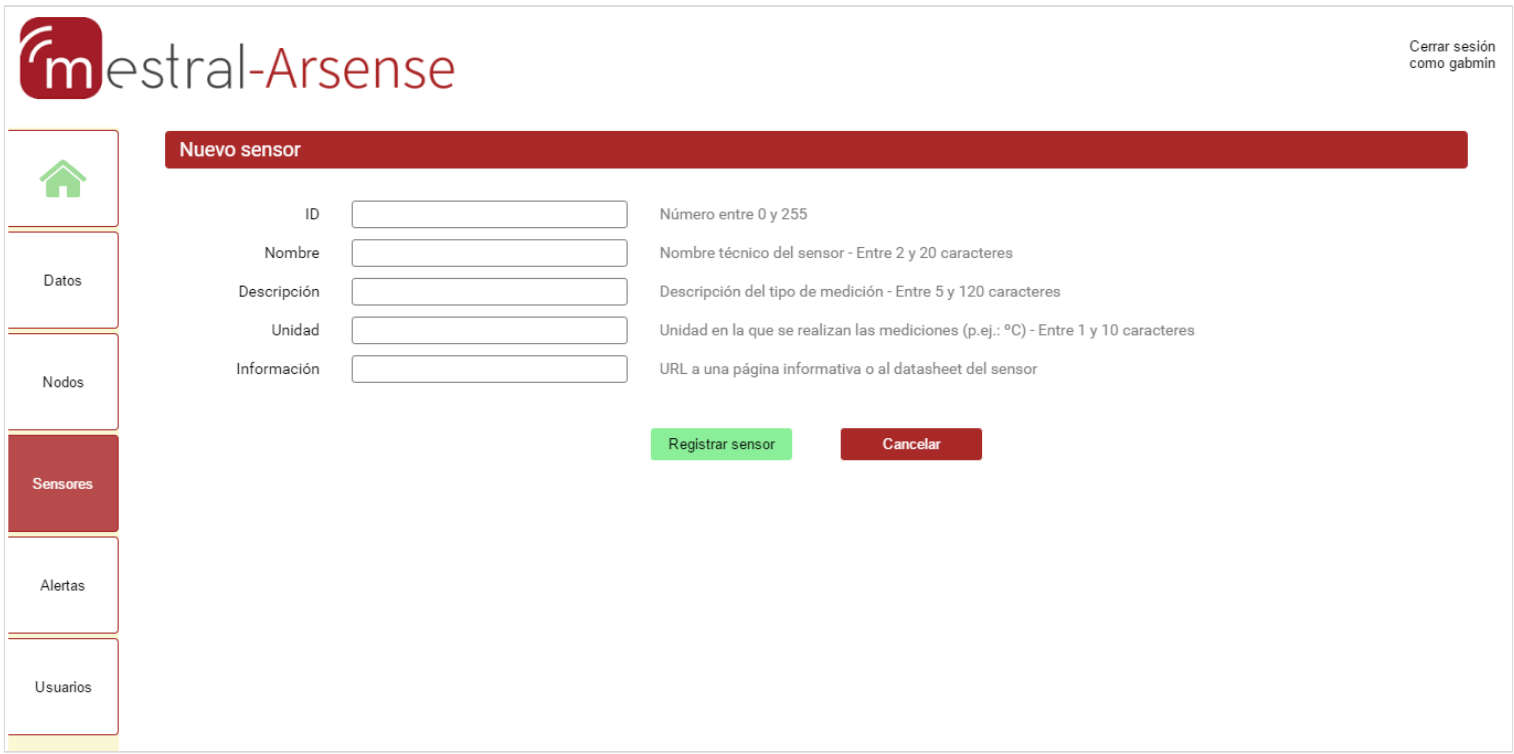


Figura 5.44: Formulario de registro/modificación de sensores

- Sección de administración de alertas:

  El campo más relevante en las entradas de la sección de alertas es la "Configuración". En este campo se muestran el nodo y los sensores que se pretenden monitorizar. Para cada sensor monitorizado se muestra el valor o rango de valores que determinan cuándo se disparará la alarma. Estos valores aparecen como una cadena de texto con el formato:

  - Min NUM <= NUM <= NUM max: si el rango de valores aptos considera un máximo, un mínimo y un valor en concreto.

  - Min NUM - NUM Max: si los valores aptos están considerados como un rango entre un mínimo y un máximo.

  - NUM: si se espera por un valor en concreto.

  En la *Figura 5.45* se muestra la sección de administración de alertas con una alerta programada a modo de ejemplo:

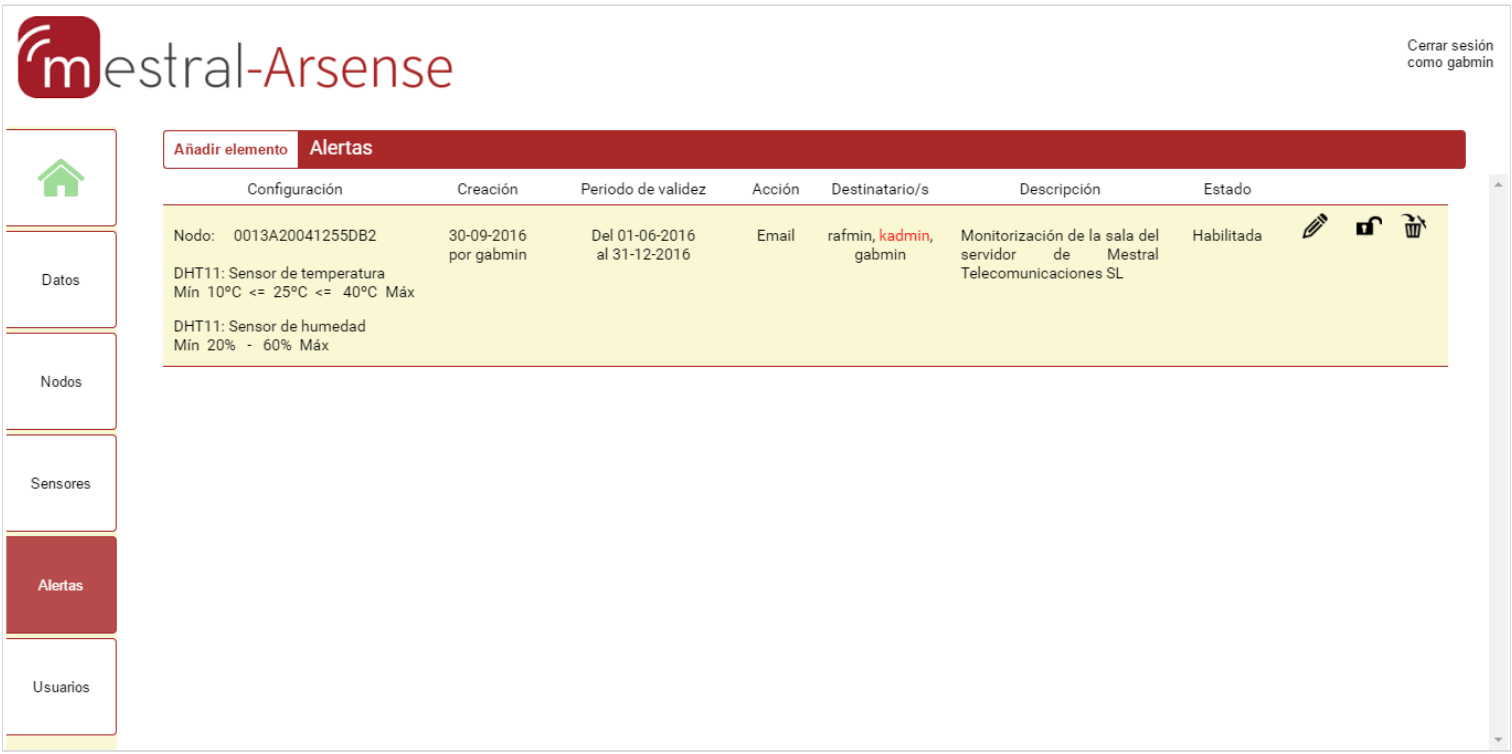


Figura 5.45: Sección de administración de alertas del *frontend*

En el formulario de registro/modificación de alertas, que puede verse en la *Figura 5.46*, aparecen dos campos clave: los usuarios a los que se notificará que se ha disparado la alerta y los sensores de los que dispone el nodo seleccionado en el desplegable, junto a otros tres campos para indicar el rango de valores a monitorizar.

Estos dos campos se comportan de igual forma que el campo de sensores del formulario de edición y creación de nodos de la *Figura 5.40*. Se omiten aquellos usuarios o sensores bloqueados o, en el caso de estar modificando una alerta que contiene a uno de estos elementos bloqueados, se muestran en rojo. Además, en el caso de los usuarios bloqueados, si se guardan los cambios, se retira el nombre de la lista de destinatarios independientemente de si se desmarca explícitamente.

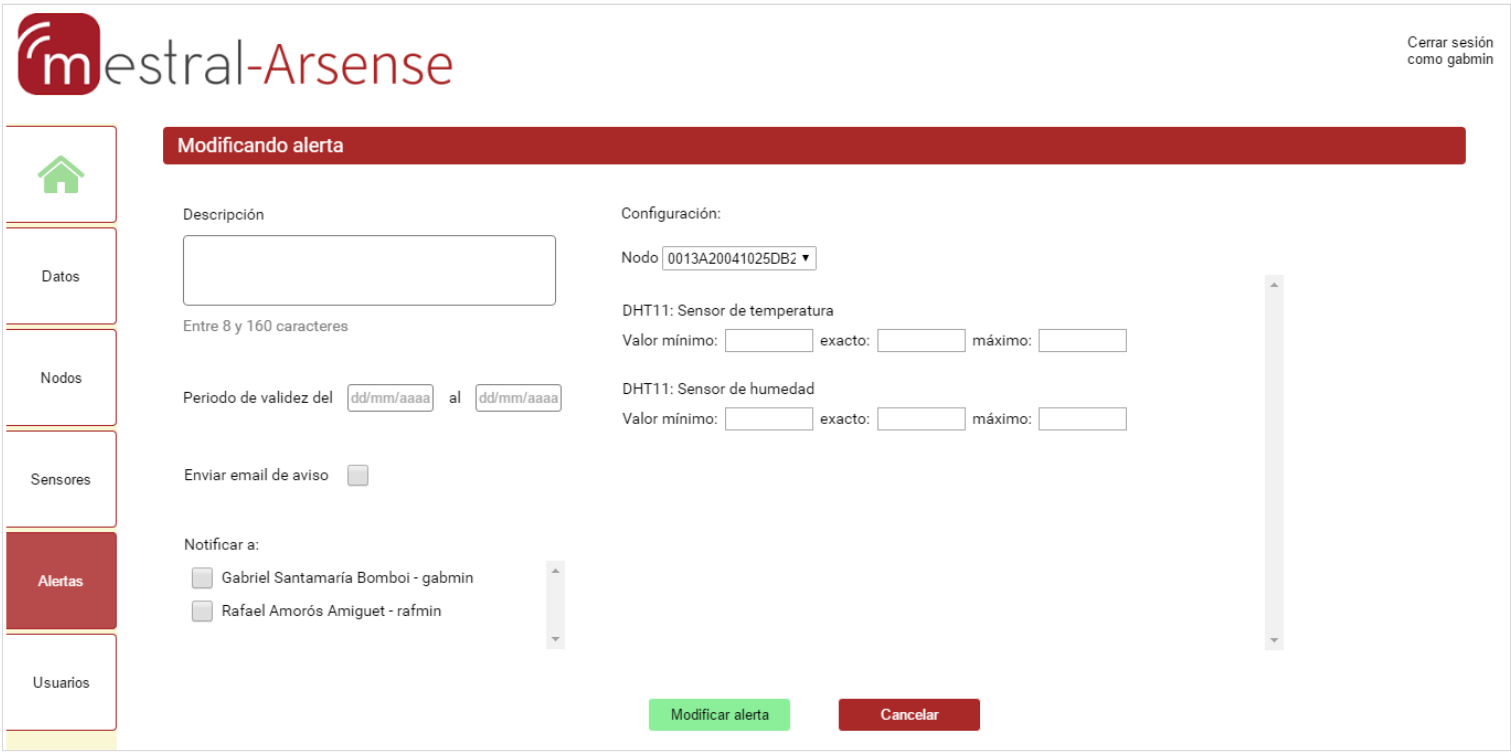


Figura 5.46: Formulario de registro/modificación de alertas

- Sección de administración de usuarios:

  Cada una de las entradas de esta sección muestra los datos de un usuario (a excepción de sus respectivas contraseñas). Puede verse una captura de la misma en la *Figura 5.47*.

  Los permisos se muestran de forma numérica como un número de cinco dígitos cada uno de los cuales se encuentra en un rango de 0 a 7.

  Cada dígito se corresponde con una de las 5 cinco secciones de la siguiente forma: el primero a la sección de datos, el segundo a la de nodos, el tercero a la de alertas, el cuarto a la sección de usuarios y el quinto y último a la de los sensores. A su vez, el valor de cada número se corresponde con la representación decimal de un número binario de tres bits. El primer bit se corresponde con los permisos para crear nuevos elementos, el segundo con el de editar y bloquear elementos ya existentes, y el tercero con el de eliminarlos de la base de datos.

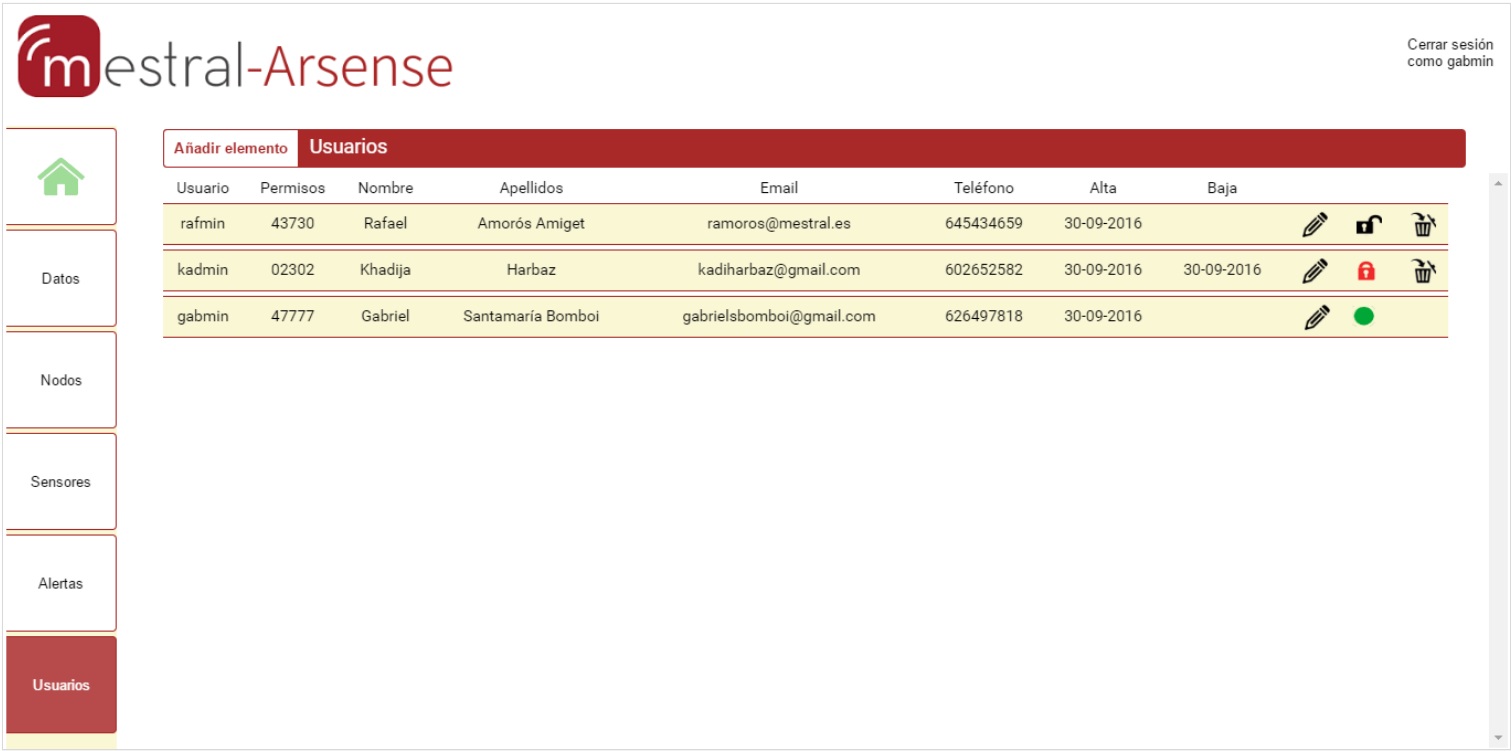


Figura 5.47: Sección de administración de usuarios del *frontend*

El usuario activo, es decir, aquel que se corresponde con la sesión actual, no muestra los botones para ser bloqueado o eliminado. En su lugar aparece un icono de color verde indicando esta situación. En la *Figura 5.48* se muestra la entrada de un usuario activo en el sistema:

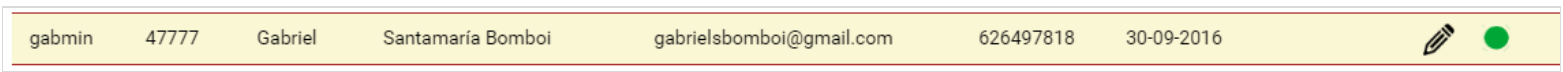


Figura 5.48: Entrada del usuario activo en el sistema

Por último, en la *Figura 5.49*, se muestra una captura del formulario diseñado para el registro y/o modificación de usuarios.

Este formulario, muestra únicamente el campo "Nombre de usuario" cuando se utiliza para registrar a un nuevo usuario en el sistema, quedando oculto cuando se trata de una modificación. Si al tratar de realizar un registro el nombre de usuarios ya consta en el sistema se mostrará una ventana emergente indicándolo.

Por otra parte, cuando se están modificando los datos de un usuario, los campos "Contraseña" y "Repetir contraseña" se utilizan para indicar una nueva contraseña.

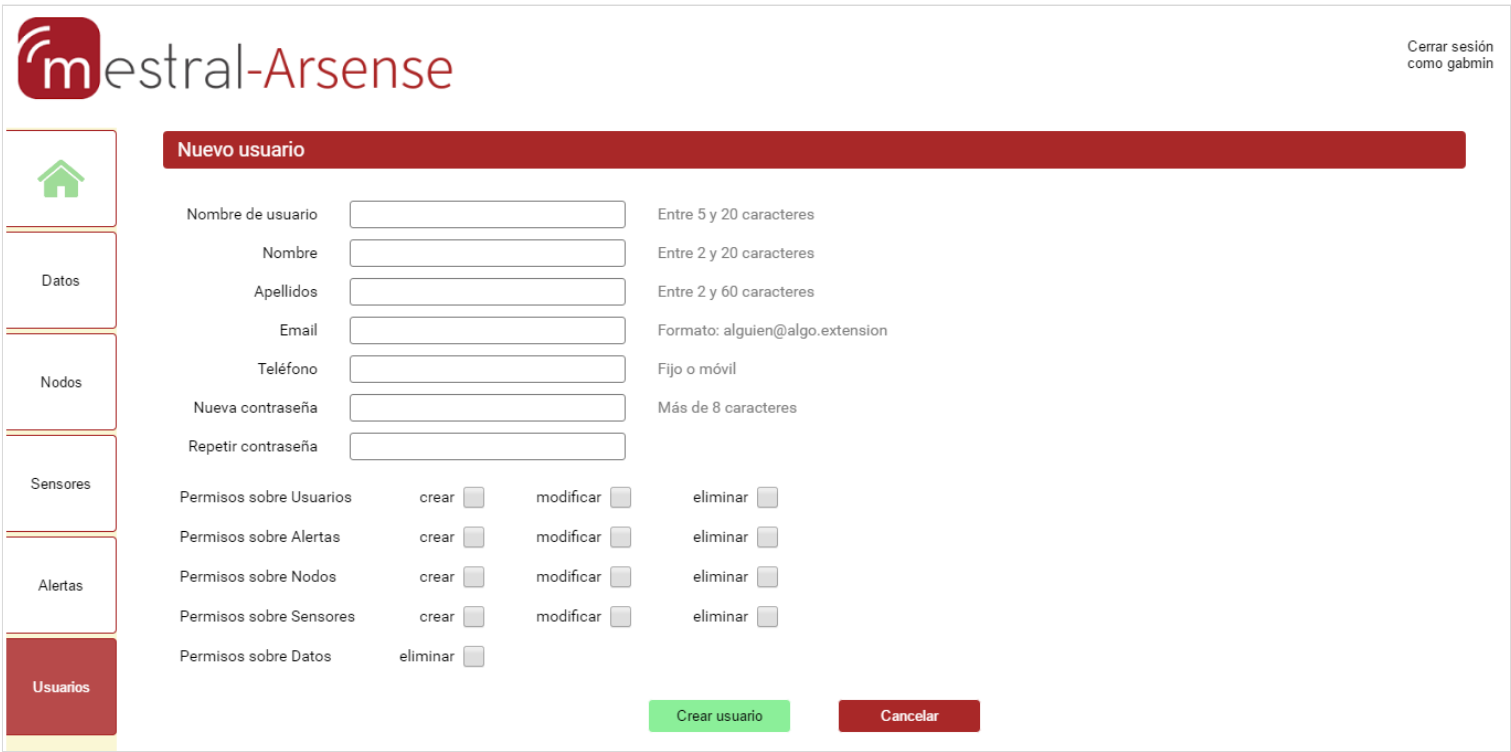


Figura 5.49: Formulario para registrar/modificar usuarios

Además, como se ha indicado en el boceto de este formulario, los permisos se muestran como casillas seleccionables independientes para cada sección y propósito (una fila por sección y una columna por tipo de permiso). El sistema se encarga posteriormente de calcular el equivalente numérico, que es el que se almacena en la base de datos.

Capítulo 6

Implementación y pruebas

En este capítulo se detallan las funcionalidades más importantes del desarrollo del sistema de monitorización del entorno y las pruebas llevadas a cabo para corroborar el funcionamiento general de todos su componentes en conjunto.

6.1. Detalles de la implementación

Las funcionalidades que se van a explicar en este apartado son las siguientes:

- Biblioteca XBee:
  - Envío de un mensaje.
  - Recepción de un mensaje.

- Software controlador de los nodos Arduino (Noduino)
  - Procesamiento de un frame.
  - Unión de un nodo a la red.
  - Envío de datos.

- Software controlador del coordinador de red (PiCo)
  - Procesamiento de un frame.

6.1.1 Biblioteca XBee: Envío de un mensaje

Para enviar un mensaje debe componerse una trama de acuerdo al formato general del modo API2. En esta trama hay unos campos que varían en función de la estructura específica del tipo de mensaje identificado por el ID de la API. Las diferentes funciones disponibles para enviar mensajes se encargan de componer la trama para la API correspondiente. Una vez que se ha compuesto el mensaje, se llama a la función *serial_tx( )*, que escapa los caracteres que puedan confundirse con símbolos de control, a la par que los transmite al medio escribiéndolos en el puerto serie.

Hay tres tipos de funciones de envío de acuerdo a las API definidas en la clase XBee. Estas son:
- *serial_ATlocal*: para comandos AT.
- *serial_tx16*: para enviar a direcciones de 16 bits.
- *serial_tx64*: para enviar a direcciones de 64 bits.

Además, existen dos funciones denominadas *serial_ack16( )* y *serial_ack64( )* que simplifican el envío de ACK componiendo un mensaje específico y utilizando la función *serial_tx16( )* o *serial_tx64( )* respectivamente para enviarlo. También hay una función denominada

serial_broadcast( ) que se comporta de forma similar a estas dos, pero mandando el mensaje a la dirección por defecto de *broadcast*, que es: 0x000000000000FFFF.

La construcción de una trama es similar en los tres tipos funciones, al margen de los campos específicos de la API. Por ello, se va a explicar de forma general el procedimiento con algunas anotaciones acerca de las particularidades de cada una de ellas.

Cálculo del tamaño total de una trama

En primer lugar antes de crear la trama se debe averiguar el tamaño total. La parte general de todos los mensaje ocupa 4 bytes, a los cuales se les debe sumar el tamaño de la parte específica de la API y el tamaño de los datos (ver *Figura 5.4*).

De la parte específica de la API (ver *Figura 5.5*) hay 2 bytes que son comunes a los tres tipos de envíos: el byte del identificador de la API y el identificador del frame. En el caso del envío de un comando se necesitan 2 bytes más para el comando, que está compuesto por dos caracteres ASCII y *n* bytes para el parámetro asociado (si lo tiene).

Para los envíos a direcciones de 16 y 64 bits (ver *Figura 5.6*) se necesita 1 byte para las opciones y entre dos y ocho para la dirección destino según se use una dirección de 16 ó 64 bits, respectivamente. Los datos tienen que ocupar al menos un byte y un máximo de 100 bytes.

En resumen, el tamaño de un frame queda de la siguiente forma:

- serial_ATlocal: 8 + tamaño del parámetro.
- serial_tx16: 9 + tamaño del mensaje.
- serial_tc16: 15 + tamaño del mensaje.

Una vez conocido el tamaño se crea un array de enteros de ocho bits sin signo (unsigned char) de ese tamaño.

```
uint8_t tam_trama = 5 + tam_API + tam_datos_enviados;
uint8_t frame[tam_trama];
```

Inserción de los campos comunes a todas las API

Los cinco primeros bytes se asignan de la misma forma a los tres tipos de envío (ver *Figura 5.4*):

`frame[0] = 0x7E;`	Es el símbolo que marca el inicio de una trama.
`frame[1] = 0x00;`	Parte más significativa del tamaño del frame. Siempre es 0 porque no se sobrepasa el tamaño máximo de 115 bytes.
`frame[2] = tam_trama;`	Parte menos significativa. Es el tamaño en bytes del frame.
`frame[3] = API_id;`	Identificador de la API: 0x08, 0x01 o 0x00 (ver *Figuras 5.5* y *5.6*).
`frame[4] = trama_id;`	Identificador de la trama.

En el envío de un comando AT, los dos siguientes bytes se corresponden con el primer y segundo carácter del comando, por lo que también se asignan directamente, y el resto son completados mediante un bucle que recorre el parámetro carácter a carácter y va insertando cada uno de ellos en la trama (ver *Figura 5.5*).

Inserción de la dirección destino en el envío de mensajes a otro nodo

Si se trata de un envío a otro nodo, a partir del sexto byte y sucesivos (ver *Figura 5.6)* se inserta la dirección destino. La asignación se realiza también directamente, pero es necesario trocear la dirección en fragmentos de 8 bytes.

Por ejemplo, en el caso de una dirección de 64 bits:

```
frame[5]  =  (direccion_nodo64.MSB >> 24) & 0xFF;
frame[6]  =  (direccion_nodo64.MSB >> 16) & 0xFF;
frame[7]  =  (direccion_nodo64.MSB >> 8)  & 0xFF;
frame[8]  =   direccion_nodo64.MSB & 0xFF;
frame[9]  =  (direccion_nodo64.LSB >> 24) & 0xFF;
frame[10] = (direccion_nodo64.LSB >> 16) & 0xFF;
frame[11] = (direccion_nodo64.LSB >> 8)  & 0xFF;
frame[12] =  direccion_nodo64.LSB & 0xFF;
```

Los bits que componen las partes alta y baja de la dirección, de 32 bits cada una, son desplazados en valores múltiplo de 8 y, a continuación, se les aplica una operación AND para quedarse con un solo byte y descartar al resto. El procedimiento para una dirección de 16 bits es mucho más sencillo ya que sólo hay que realizar este desplazamiento una vez.

Los siguientes campos son el byte de opciones y los datos. La forma de insertarlos es análoga a la inserción de un parámetro en un comando AT: un bucle que recorre cada carácter del array de caracteres de datos y lo inserta en la trama:

```
while (indice  < tam_datos_enviar) { frame[14 + indice ] = msg[indice++]; }
```

Cálculo e inclusión del checksum

Por último, tanto para el envío de datos como de comandos AT, se añade el byte correspondiente al checksum. Este valor se calcula restando a 0xFF los bits menos significativos de la suma de todos los símbolos correspondientes a la parte específica de la API.

Para agilizar el proceso, y aunque no se ha mostrado, cada vez que se inserta un carácter en la trama se le suma también a la variable del checksum.

La asignación de este último campo resulta así:

```
frame[24 + tam_datos_enviar] = 0xFF - (checksum & 0xFF);
```

Envío mediante la función serial_tx( )

A continuación, con la trama construida, se llama a la función *serial_tx( )*, que recorre todos los caracteres del array mediante un bucle, los escapa si es necesario y los escribe en el puerto serie.

Para escapar un carácter se comprueba (a excepción del byte de inicio) si es igual a:

- 0x7E: byte de inicio.
- 0x7D: byte que indica carácter escapado.
- 0x13 o 0x11: usados por XBee para controlar el flujo de datos.

En caso afirmativo se envía primero el carácter 0x7D, que le indica al receptor que el siguiente carácter va escapado, y el símbolo se manda a continuación aplicando una operación XOR por 0x20:

```
for (uint8_t i = 1; i < frameTam; i++) {
      uint8_t c = frame[i];

      if (c == 0x7D || c == 0x7E || c == 0x11 || c == 0x13) {
            puerto_serial.escribir(0x7D);
            puerto_serial.escribir((c ^ 0x20));
      }
      else {
            puerto_serial.escribir(c);
      }
 }
```

En este punto, el mensaje ya se encuentra camino del destinatario. Es importante que el checksum esté correctamente calculado, de otro modo el mensaje no saldrá del módulo XBee.

6.1.2 Biblioteca XBee: Recepción de un mensaje

La recepción de un mensaje se realiza específicamente con la función *serial_tx( )*, que se encarga de detectar cuándo se está recibiendo un frame y de comprobar si es válido, e incluye también la llamada a las funciones *extraerDatosFrame( )* y *procesarFrame( ) d*e los programas controladores de los nodos y el coordinador. La función *extraerDatosFrame( )* se encarga de procesar un frame para extraer todos los campos de interés y la función *procesarFrame( )* de determinar la respuesta a dicho mensaje.

La función *serial_tx( )* finaliza cuando recibe un frame indicando si es correcto o no y devolviendo un objeto del tipo *DatosFrame*. Por tanto, para que un nodo se mantenga a la escucha, se ha implementado una función complementaria en el software controlador del nodo denominada *escuchar( )*, la cual realiza llamadas consecutivas a esta función durante el periodo de tiempo indicado.

Función serial_tx( )

Se inicia un bucle que se ejecuta mientras no se sobrepase el *timeout* indicado como argumento. Dentro de este bucle se comprueba si hay algún caracter esperando en el puerto serie mediante la función *disponible( )* de *SerialManager.*

```
while (millis() - timer <= timeout) {
      if (puerto_serial.disponible() > 0) { (...) }}
```

La primera comprobación, tras constatar que han llegado datos al módulo XBee, consiste en comprobar si previamente se había detectado una trama entrante. Esta comprobación, mediante la variable *frameEsperando*, sirve para evitar que dos mensajes que hayan llegado a continuación uno del otro se procesen como uno solo. De no ser así se perdería el segundo, ya que quedaría fuera del procesamiento al encontrarse más allá del tamaño especificado por la primera trama.

El valor de la variable *frameEsperando* es establecido más adelante.
Si *frameEsperando* es *true*, el primer byte de la trama de entrada se pone a 0x7E y se cambia la variable *frameEntrante* a *true*.

```
if (frameEsperando == true) {
        inBuff[inTam++] = 0x7E;
        frameEntrante = true;
}
```

En este punto se lee un caracter del puerto serie: `uint8_t rxc = puerto_serial.leer();`

Si el carácter recibido es el byte de inicio de una trama (0x7E) se comprueba el número de bytes de entrada almacenados hasta el momento. Si es 0 significa que se trata de una nueva trama entrante y la variable *frameEntrante* pasa a *true*. En caso contrario significa que se ha llegado al final del anterior frame y que se ha pasado a procesar una trama distinta. En este último caso la variable *frameEsperando* pasa a *true*, el proceso de recepción termina y se continúa procesando la primera trama.

```
if (rxc == 0x7E) {
        if (inTam == 0) {
        frameEntrante = true;
        } else {
             frameEsperando = true;
        break;
        }
}
```

Si el proceso continúa normalmente, se comprueba la variable *frameEntrante* para determinar si se están recibiendo datos y, en caso afirmativo, se procede a procesar el carácter leído.

En primer lugar se comprueba si dicho carácter es igual a 0x7D (que como se ha comentado indica que el siguiente caracter ha sido escapado). En caso afirmativo se espera por el siguiente carácter y una vez recibido, si no es que se encuentra ya en el búffer, se le aplica una operación XOR por 0x20.

Finalmente se guarda el carácter en el array que almacena la trama entrante, se incrementa el índice para que apunte a la siguiente posición y la variable *frameEsperando* pasa a false.

```
if (frameEntrante == true) {
      if (rxc == 0x7D) {
            while (puerto_serial.disponible( ) == 0);
            rxc = puerto_serial.leer( ) ^ 0x20;
      }

      inBuff[inTam] = rxc;
      ++inTam;
      frameEsperando = false;
}
```

Una vez leída la trama completa se considera válida si:

- El byte 0 es 0x7E.
- El tamaño indicado en el byte 3 se corresponde con el total de bytes leídos.
- El identificador de la API es 0x80, 0x81 o 0x88.
- Se ha enviado al menos un byte de datos en los frames con API 0x80 (dirección 64 bits) y 0x81 (dirección 16 bits).

Función extraerDatosFrame( )

Una vez recibida la trama y comprobado que es válida, el siguiente paso del proceso es la extracción de los datos mediante la función *extraerDatosFrame( )*. La función *serial_tx( )* espera la respuesta de ésta en forma de objeto del tipo *DatosFrame*, que es lo que finalmente se devuelve por referencia.

Si se trata de un mensaje de tipo ACK o respuesta a un comando AT, se actualizan las listas de estado asociadas a estos elementos. Estas listas pueden ser consultadas por los programas controladores de los nodos y del coordinador de la red para comprobar si ya tienen la respuesta a un comando o ha llegado el ACK que esperan:

```
if (drx.APIid == 0x88) {
	uint8_t comando[] = {drx.comando[0], drx.comando[1]};
	upStatus_rcmdAT(comando, drx.status);
}
if (drx.datos[0] == ACK) upStatus_ack(drx.datos[1], drx.datos[2]);
```

La función *extraerDatosFrame( )* comprueba en primer lugar el identificador de la API de la trama recibida y lo guarda en el objeto *DatosFrame*.

```
datosFrame.APIid = frame[3];
if (datosFrame.APIid == 0x80 || datosFrame.APIid == 0x81) {  (...)  }
else if (datosFrame.APIid == 0x88) {  (...)  }
```

Si el valor de este identificador es 0x88 se trata de la respuesta a un comando AT local (ver *Figura 5.5 inferior*). En este caso los bytes correspondientes a los dos dígitos del comando que produjo dicha respuesta se obtienen de los bytes 6 y 7, y el estado que indica si se procesó correctamente o no, del byte 8. Para obtener el tamaño de la respuesta, si la hay, se resta al tamaño total del frame el de los campos obligatorios y a continuación se recorre dicha cantidad de bytes a partir del byte 9 de la trama, guardando cada símbolo en un array, obteniendo así el valor devuelto por el módulo.

```
if (drx.APIid == 0x88) {
	datosFrame.comando[0] = frame[5];
	datosFrame.comando[1] = frame[6];
	datosFrame.status = frame[7];
	datosFrame.datosTam = frameTam - 9;
	datosFrame.datos = (uint8_t*) malloc(datosFrame.datosTam *
	sizeof(uint8_t));

	while (datosFrame.datosTam > 0 && c < datosFrame.datosTam) {
		datosFrame.datos[c] = frame[8 + c];
		c++;
	} }
```

Si el identificador de la API es 0x80 o 0x81 se trata de un mensaje proveniente de otro nodo (ver *Figura 5.6 inferior*). En este caso se identifica en primer lugar si es 0x81 (dirección origen 64 bits) o 0x80 (dirección origen 16 bits). La función implementa la extracción de los campos como si se tratase de un mensaje de una dirección de 16 bits, pero utiliza una variable denominada *desplazamiento* que toma el valor 6 si se trata de una dirección de 64 bits. De este modo se evita tener que repetir el procedimiento cambiando únicamente el índice a consultar.

El primer campo que se extrae es la dirección de origen, que ocupa los bytes 5 y 6 para las direcciones de 16 bits o desde el 5 hasta el 12 para las direcciones 64 bits, le siguen el RSSI (byte 7 + *desplazamiento*) y a continuación el byte de opciones (byte 8 + *desplazamiento*).

```
datosFrame.rssi = frame[6 + desplazamiento];
datosFrame.op = frame[7 + desplazamiento];
```

La dirección, tal y como se explica en el apartado 6.1.1, se encuentra fragmentada en trozos de 1 byte, por lo que es necesario volverla a unir. Para una dirección de 64 bits se hace de la siguiente forma:

Parte más significativa (32 bits):

```
datosFrame.origen64.DH = (uint32_t(frame[4]) << 24) + (uint32_t(frame[5]) << 16) +
(uint16_t(frame[6]) << 8) + frame[7];
```

Parte menos significativa (32 bits):

```
datosFrame.origen64.DL = (uint32_t(frame[8]) << 24) + (uint32_t(frame[9]) << 16) +
(uint16_t(frame[10]) << 8) + frame[11];
```

Para una dirección de 16 bits se procede de igual forma, pero concatenando únicamente los dos bytes que la forman y almacenándola en el campo *origen16* del objeto *datosFrame*.

El tamaño de los datos se obtiene restando al byte 3 de la trama el valor (9 + desplazamiento), que es el total de campos de la API. A continuación los datos se extraen con un bucle que recorre la trama desde el byte 8 + desplazamiento hasta el tamaño calculado.

```
datosFrame.datosTam = frameTam - (9 + desplazamiento);

while (c < datosFrame.datosTam) {
      datosFrame.datos[c] = frame[(8 + desplazamiento) + c];
      ++c;
}
```

Una vez extraídos los campos se devuelve el objeto de tipo *DatosFrame*.

De esta forma la gestión de un mensaje entrante es más simple, ya que el software controlador de los nodos o del coordinador no necesita preocuparse del tipo de API o de los bytes correspondientes a cada campo.

A continuación, se detalla el procesamiento de un frame en los nodos Arduino y en el coordinador de la red.

6.1.3 Noduino: Procesamiento de un frame

Cuando un nodo Arduino recibe un mensaje y obtiene el objeto de tipo *DatosFrame,* llama a la función *procesarFrame( )* a la que le pasa dicho objeto como parámetro.

El primer paso que lleva a cabo esta función es la obtención de la cabecera del tipo de mensaje, que se corresponde con el byte 0 de los datos del mensaje.

```
uint8_t cabecera = datosFrame.datos[0];
```

Si este valor no se corresponde con ninguno de los contemplados se devuelve *false* y se considera un mensaje erróneo o que no debería haberse recibido. En caso contrario hay 6 posibles acciones a llevar a cabo, es decir, que un nodo Arduino puede recibir 6 tipos de mensajes distintos (tal y como se explica en el *apartado 5.1.2* sobre el formato de las comunicaciones):

- Mensaje con destino al coordinador de la red (ver cabecera 0x31 en la *Figura 5.7*): el nodo reenvía el mensaje a su coordinador, que puede ser el coordinador de la red u otro nodo si está siendo enrutado.

```
xbee.serial_tx64(direccionCoord, datosFrame.datos, datosFrame.datosTam, 0x00, 0x00);
```

- Mensaje con destino un nodo de la red (ver cabecera 0x33 en la *Figura 5.9*): en primer lugar se extrae la dirección situada en los bytes siguientes a la cabecera del mensaje. A continuación, se extrae de los datos del frame y, si el carácter que sigue indica fin de la ruta ('#') se extrae también la cabecera que indica que es un mensaje con destino a un nodo. Finalmente es reenviado a la dirección extraída en primer lugar.

```
Direccion64bits destino;

destino.DH = (uint32_t(drx.datos[1]) << 24) + (uint32_t(drx.datos[2]) <<
16) + (uint16_t(drx.datos[3]) << 8) + drx.datos[4];

destino.DL = (uint32_t(drx.datos[5]) << 24) + (uint32_t(drx.datos[6]) <<
16) + (uint16_t(drx.datos[7]) << 8) + drx.datos[8];

uint8_t uc = 0;

if (drx.datos[9] == '#') {
       while (uc < drx.datosTam) {
              drx.datos[uc] = drx.datos[uc + 9];
              drx.datosTam = drx.datosTam - 9;
       }
}
else {
       while (uc < drx.datosTam) {
              drx.datos[uc + 1] = drx.datos[uc + 9];
              drx.datosTam = drx.datosTam - 8;
       }
}

xbee.serial_tx64(destino, drx.datos, drx.datosTam, 0x00, 0x00);
```

- Mensaje solicitando información de la conexión con el coordinador de la red (ver cabecera 0x35 en la *Figura 5.11*): si el nodo tiene acceso a un coordinador envía sus datos de conexión a la dirección origen del mensaje recibido.

```
if (direccionCoord.DH != 0x0000000 && direccionCoord.DL != 0x0000000) {
       msg[] = {xbee.DATOS_COORDINADOR, rssiCoord, 0, enrutDirecto};
       xbee.serial_tx64(datosFrame.origen64, msg, 4, 0x00, 0x00);
}
```

- Mensaje con información de conexión al coordinador de red de un nodo vecino (ver cabecera 0x36 en la *Figura 5.12*): cuando se reciben los datos de conexión al coordinador de un nodo vecino, el objeto *DatosFrame* se almacena en una cola FIFO para ser gestionado en otro momento. Si esta cola está llena se retira el último elemento que llegó para dejar sitio al nuevo.

```
if (cola_vecinos.count() > MAX_COLA_TAM)
       cola_vecinos.pop_last();

cola_vecinos.push(datosFrame);
```

- Mensaje preguntando al nodo si sigue activo (ver cabecera 0x38 en la *Figura 5.14*): se extrae el identificador del ACK que espera el emisor, y que se corresponde con el byte 1 de los datos y se envía un mensaje de confirmación con la función *serial_ack64( )* indicando como estado '1' (OK).

```
uint8_t ack_id =  datosFrame.datos[1];
xbee.serial_ack64(datosFrame.origen64, ack_id, OK);
```

- Mensaje informando que ha sido escogido como coordinador (ver cabecera 0x37 en la *Figura 5.*13): si un nodo recibe este mensaje significa que el nodo que lo está enviando necesita que sea su enrutador hacia el coordinador de la red, ya que no ha obtenido respuesta de este último.

  Si el nodo sigue teniendo acceso a su coordinador compone un mensaje con la cabecera CONFIRMA_NODO_REMOTO en el que incluye la dirección del nodo de 64 bits a modo de identificador, además de un identificador de ACK.

```
t msg[] = {xbee.CONFIRMA_NODO_REMOTO,
     (drx.origen64.DH >> 24) & 0xFF, (drx.origen64.DH >> 16) & 0xFF,
     (drx.origen64.DH  >> 8) & 0xFF, drx.origen64.DH & 0xFF,
     (drx.origen64.DL >> 24) & 0xFF, (drx.origen64.DL >> 16) & 0xFF,
     (drx.origen64.DL >> 8) & 0xFF, drx.origen64.DL & 0xFF, ack_id
 };
```

  El valor de la variable *ack_id* es el mismo identificador de ACK que utiliza el nodo emisor de origen.

A continuación, envía el mensaje al coordinador.

```
txCoordinadorRed(msg, 10, 0x00, 0x00);
```

Y espera también por una confirmación de que el nodo es válido para ponerse en modo coordinador (es decir, que no duerme).

```
escuchar(60000);
if (xbee.status_ack(ack_id, true)) {
        enrutador += 1;
}
```

Por último, el coordinador de la red envía un ACK directamente al solicitante.

Los ACK y las respuestas de comandos AT no son incluidos en el grupo de mensajes que puede procesar el nodo ya que son gestionadas "internamente" por la biblioteca XBee. Aunque es posible consultar el estado de un comando AT a través de las funciones *status_ack( )* y *status_rcmdAT( )*.

6.1.4 Noduino: Unión de un nodo a la red

La unión de un nodo a la red está relacionada con la búsqueda del coordinador. Cuando un nodo obtiene acceso al coordinador de la red es marcado como activado en la base de datos del servidor.

Este proceso se realiza en dos fases:

- Búsqueda de nodos vecinos: se envía un broadcast con un mensaje cuyo contenido es la cabecera QUIERO_DATOS_COORDINADOR (0x35).

  ```
  uint8_t msg[] = {xbee.QUIERO_DATOS_COORDINADOR};
  xbee.serial_broadcast(msg, 1, 0x00);
  ```

  A continuación, se llama a la función *escuchar( ),* la cual recibe mensajes durante el tiempo pasado como argumento *timeout*. Como se ha explicado en el *Apartado 6.1.3* referente al procesamiento de un frame en los nodos Arduino, cada vez que el nodo recibe un mensaje con los datos de conexión al coordinador de un vecino los almacena en una cola FIFO.

  Una vez finalizada la búsqueda de vecinos entra en juego la función *actualizarCoordinador( ).*

- Actualización del coordinador: en primer lugar se crean variables para almacenar los datos del candidato a coordinador temporalmente. A continuación, se recorre la cola FIFO de vecinos y se comparan los datos recibidos con los de las variables temporales y, si el coordinador es el mejor candidato, se actualizan sus valores.

  ```
  if (v.datos[1] == '1' || (enrutDirecto_temp == '0' && v.datos[3] ==
  '1') || (enrutDirecto_temp == '1' && rssiCoord_temp < v.datos[2])) {
        (...) // se guarda la configuración temporal del coordinador
  }
  ```

Para considerar a un vecino como mejor candidato que otro se debe cumplir una de las siguientes condiciones:

- El coordinador actual no sea el coordinador principal.
- El nodo vecino que ha respondido es el coordinador principal.
- El actual coordinador no sea directo y el nuevo sí.
- El coordinador actual también ve al coordinador de la red directamente pero el nuevo está más cerca.

Una vez finalizado el repaso de vecinos se envía un mensaje comunicando al nodo escogido que ahora es el coordinador del nodo y se espera por un ACK. Si la respuesta a este ACK es afirmativa, la información del coordinador se copia a las variables de configuración del coordinador y el nodo pasa a estado activo.

```
msg[ ] = {xbee.SELECCIONADO_COMO_ENRUTADOR, ackid};
xbee.serial_tx64(direccionCoord_temp, msg, 2, 0x00, 0x00);

escuchar(60000);

if (xbee.status_ack(ackid, true)) {
      (...) // se guarda la configuración del nuevo coordinador
}
```

6.1.5 Noduino: Envío de datos

Para el envío de datos de un nodo al coordinador se utilizan las funciones *txDatos( ),* que compone el mensaje a enviar, y *leerSensores( ),* del módulo Sense, que recopila los datos de los sensores y genera un array de caracteres con el formato: ID sensor, valor.

La generación del mensaje de datos en el módulo Sense se realiza de la siguiente forma:

- Desde la función *leerSensores( )* se realiza la lectura de los sensores utilizando la biblioteca específica de cada uno.

  Por ejemplo, la lectura de la temperatura mediante el sensor DHT11 se realiza de la siguiente forma:

  ```
  uint32_t valor = dht.readTemperature();
  ```

  En el byte 0 se añade la cabecera para identificarlo como un mensaje de datos:

  ```
  lectura_prev[lecturaTam++] = xbee.DATOS();
  ```

  Si un sensor está habilitado (*habilitado = true*) se añade su índice al array de datos y a continuación el valor fragmentado en bloques de 8 bits. Recordar que, tal y como se indica en la *Figura 5.15*, los valores de los sensores utilizan 4 bytes.

Así pues, para el sensor DHT11 el procedimiento descrito se resume como sigue:

```
if (habilitado_1) {
      lectura_prev[lecturaTam++] = id_sensor1;

      uint32_t valor = dht.readTemperature();
      valor = isnan(valor) ? 0 : valor;
      lectura_prev[lecturaTam++] = (valor >> 24) & 0xFF;
      lectura_prev[lecturaTam++] = (valor >> 16) & 0xFF;
      lectura_prev[lecturaTam++] = (valor >> 8) & 0xFF;
      lectura_prev[lecturaTam++] =  valor & 0xFF;
  }
```

No se añaden separadores ya que la lectura en el coordinador se realiza por bloques: 1 byte para el identificador del sensor, 4 bytes para el valor leído.

Por último, se redimensiona el array de caracteres para eliminar el espacio sobrante.

```
      uint8_t* lectura = (uint8_t*) malloc(lecturaTam *
      sizeof(uint8_t));
      while (uc < lecturaTam) {
            lectura[uc] = lectura_prev[uc];
            ++uc;
      }
      free(lectura_prev);
```

- Desde la función *txDatos( )* se realiza el envío de los datos: esta función, si tiene acceso a un coordinador, invoca a *leerSensores( )* y envía el array de bytes obtenido, que es el mensaje a enviar, al coordinador de la red.

```
lecturaSensores = leerSensores(lecturaTam);
if (lecturaSensores != NULL) {
      txCoordinadorRed(lecturaSensores, lecturaTam, 0x00, 0x00);
}
```

6.1.6 PiCo: Procesamiento de un frame

Cuando el coordinador de la red recibe un mensaje y obtiene el objeto *DatosFrame* llama a la función *procesarFrame( )* a la que le pasa dicho objeto como parámetro.

Los primeros pasos que realiza esta función consisten en comprobar si el identificador de la API se corresponde con un mensaje de datos de una dirección de 64 bits y tomar una marca de tiempo para ser usada en el log y en el registro de datos.

```
if (APIid == 0x80) {
      uint8_t* tiempo = timestamp();
      (...)
}
```

A continuación obtiene la cabecera del tipo de mensaje, que se corresponde con el byte 0 de los datos del mensaje.

```
uint8_t cabecera = datosFrame.datos[0];
```

Si este valor no se corresponde con ninguno de los contemplados se muestra un mensaje indicando que se ha recibido un mensaje que no se ha podido procesar y en caso contrario se procede a procesarlo.

Para todos los mensajes entrantes, y aunque no se especifique en cada apartado, se comprueba que las direcciones indicadas, tanto de origen como empaquetadas, se corresponden con nodos registrados y activados, cotejando con la base de datos a través de la función *comprobarNodo( ).*

El coordinador de la red puede gestionar 5 mensajes diferentes:

- Mensaje desde un nodo lejano que ha sido enrutado (ver cabecera 0x36 en la *Figura 5.12*): cuando se recibe un mensaje con esta cabecera significa que se trata de un mensaje que no procede del nodo emisor. El procedimiento para procesarlo es desempaquetar la dirección del nodo emisor real y establecerla como origen del objeto *DatosFrame*. A continuación se extrae de los datos la cabecera y dicha dirección, y la función se llama a sí misma con el objeto *DatosFrame* modificado, el cual tendrá en el byte 0 la cabecera original y podrá ser procesado.

```
Direccion64bits idnodo;

idnodo.DH = (uint32_t(datosFrame.datos[1]) << 24) +
(uint32_t(datosFrame.datos[2]) << 16) +
(uint16_t(datosFrame.datos[3]) << 8) + datosFrame.datos[4];

idnodo.DL = (uint32_t(datosFrame.datos[5]) << 24) +
(uint32_t(datosFrame.datos[6]) << 16) +
(uint16_t(datosFrame.datos[7]) << 8) + datosFrame.datos[8];


datosFrame.origen64 = idnodo;

for (uint8_t uc = 9; uc < datosFrame.datosTam; uc++) {
      msg[uc - 9] = datosFrame.datos[uc];
}

datosFramedatosFrame.datos = msg;
datosFrame.datosTam = datosFrame.datosTam - 9;

procesarFrame(datosFrame);
```

- Mensaje solicitando información de la conexión con el coordinador (ver cabecera 0x35 en la *Figura 5.11*): se envían los datos del coordinador al nodo solicitante según el formato del tipo de mensaje correspondiente a la cabecera 0x36 de la *Figura 5.12*.

```
uint8_t msg[] = {xbee.DATOS_COORDINADOR, 255, '1', '1'};
xbee.serial_tx64(datosFrame.origen64, msg, 4, 0x00, 0x00);
```

- Mensaje informando que ha sido escogido como coordinador (ver cabecera 0x37 en la *Figura 5.13*): se devuelve un ACK con la confirmación si el nodo existe en la base de datos. En este caso cuando se invoca a la función *comprobarNodo( )* se especifica en el segundo parámetro *true* para que sea marcado como validado.

```
xbee.serial_ack64(drx.origen64, datosFrame.datos[1], OK);
```

- Mensaje de un nodo remoto solicitando ser confirmado en la red (ver cabecera 0x32 en la *Figura 5.8*): en primer lugar se extraen la dirección origen (bytes 2 al 9) y el ID de confirmación (byte 10). La dirección del enrutador es la dirección origen del mensaje ya que aunque haya sido enrutado es restaurada al ser desempaquetado (ver *Mensaje désde un nodo lejano que ha sido rutado*). A continuación se obtiene la ruta al nodo a través de la función *getRuta()* y se realiza un procedimiento similar al del *Mensaje informando de que ha sido escogido como coordinador*, pero ahora el mensaje se empaqueta con la cabecera DESTINO_NODO_RED (ver cabecera 0x33 de la *Figura 5.9*). También se envía confirmación al coordinador.

```
uint8_t ackid = datosFrame.datos[9];
msg[ ] = {xbee.DESTINO_NODO_RED, [RUTA]. '#', xbee.ACK, ackid, OK};
xbee.serial_tx64(RUTA[0], msg, tam_msg, 0x00, 0x00);
msg[ ] = {xbee.DESTINO_NODO_RED, [RUTA2]. '#', xbee.ACK, ackid, OK};
xbee.serial_tx64(RUTA[0], msg2, tam_msg, 0x00, 0x00);
```

- Mensaje con datos de sensores (ver cabecera 0x39 en la *Figura 5.15*): se recorren los datos recibidos a partir del byte 1. Se considera que el siguiente byte es el identificador del nodo y los 4 bytes siguientes son el valor recibido en hexadecimal. Este proceso se realiza hasta que se leen todos lo datos.

```
while (drx.datos[uc] != '#') {
       sensor = 0; valor = 0;
       sensor = drx.datos[uc++];
       valor = (uint32_t(drx.datos[uc++]) << 24) +
       (uint32_t(drx.datos[uc++]) << 16) + (uint16_t(drx.datos[uc++])
       << 8) + drx.datos[uc++];
       sprintf(linea, "I:%016llX S:%c V:%.2f T:%s",
       drx.origen64.get64(), (char)
       sensor, (float) valor,  tiempo); }
```

Los datos se almacenan en un fichero utilizando la función *log( )* al que se le indica la línea a imprimir, si se debe añadir o no una marca de tiempo al final y el fichero destino. El formato de una línea del fichero de datos es el siguiente:

```
ID_NODO | ID_SENSOR | VALOR | MARCA_DE_TIEMPO
```

Estos campos están delimitados por los caracteres 'I', 'S', 'V' y 'T' respectivamente, separados por espacios en blanco, tal y como se muestra en el siguiente ejemplo:

I 12345678 **S** 01 **V** 18 **M** 01-12-2016 16:35

De forma general cada evento, mensaje o acción relevante se muestra resumida por pantalla y se almacena en el fichero de *log* utilizando la función *log( )*.

6.2. Verificación y validación

En este proyecto se tiene como objetivo principal obtener datos de sensores por lo que es necesario validar que los nodos de la red son capaces de unirse a la red y de recolectar información para hacerla llegar al coordinador, el cual debe poder transmitir dicha información a la base de datos.

Otra cuestión importante en la red es la capacidad de enrutado, por lo que también se debe comprobar si un nodo sin acceso al coordinador es capaz de buscar a otro nodo que asuma dicho papel y, en este último caso, que los datos transmitidos por dicho sensor también lleguen a la base de datos.

También se debe corroborar que el *frontend* permite gestionar usuarios, nodos y alertas creándolas, modificándolas y eliminándolas.

A continuación, se pasa a describir las verificaciones realizadas para cada uno de los objetivos planteados en el desarrollo del proyecto.

Proceso de verificación y validación sin acceso a la base de datos

En primer lugar, se comprueba que los nodos y el coordinador de la red pueden comunicarse, es decir, que el protocolo diseñado para unirse a la red y enrutar datos funciona. Se ha realizado una monitorización del log del coordinador sin el acceso a la base de datos activo (modificando su comportamiento para que considere a cualquier nodo válido).

Además, para comprobar la función de enrutado entre nodos, se ha modificado el software controlador del coordinador para que rechace la conexión directa del nodo cuyo identificador es 0013A20041255DBAA.

A continuación, en la *Figura 6.1*, se muestra y explica la salida por pantalla del log del coordinador de la red visualizado a través de un terminal en la Raspberry Pi. En dicho log se muestra cuando el coordinador es iniciado, la unión de un nodo a la red de forma directa y través de un nodo vecino, y la recepción de datos.

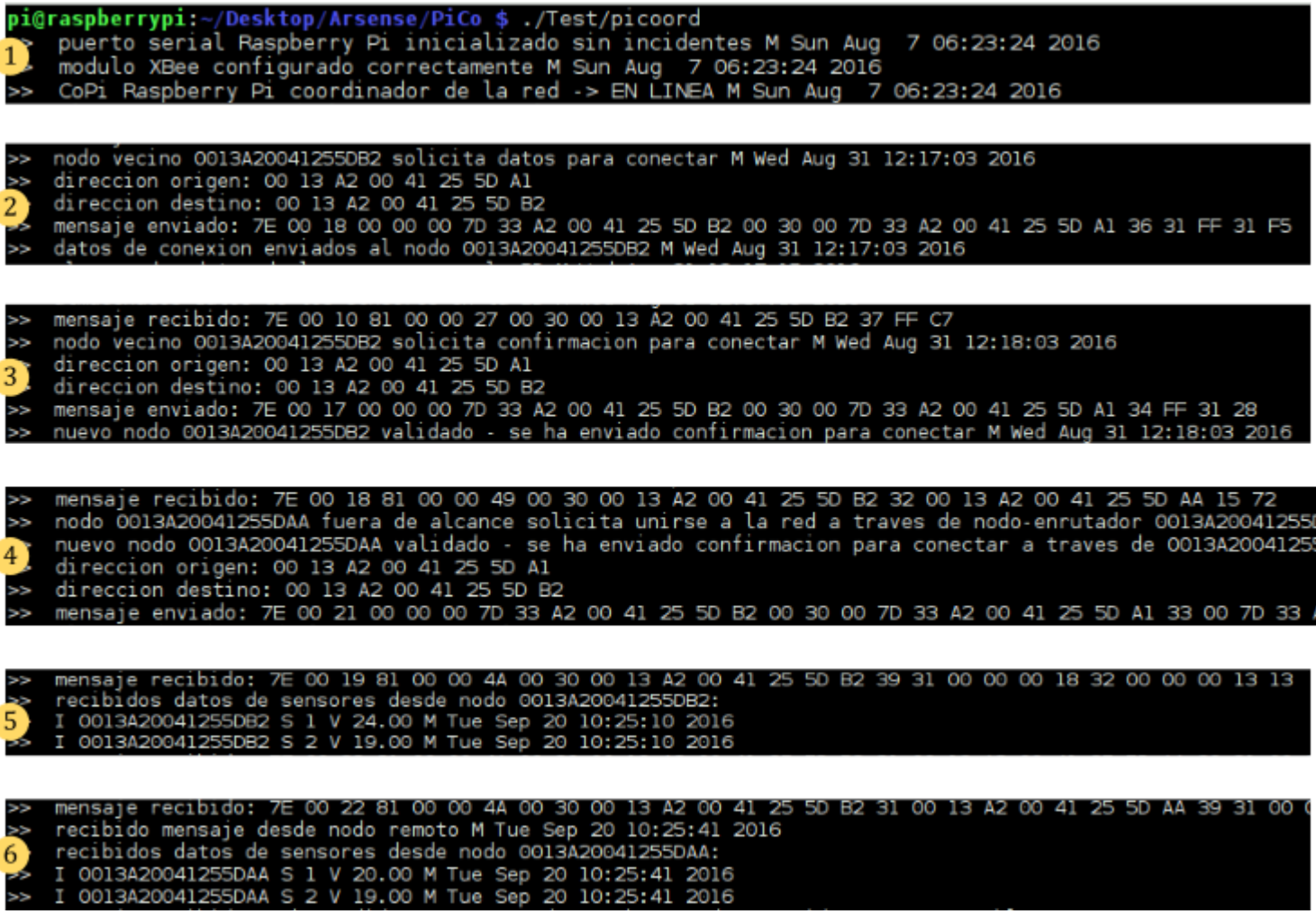


Figura 6.1: Comprobación de las comunicaciones a través del log del coordinador

1. Inicialización del coordinador: muestra mensajes indicando que la configuración del módulo XBee, la inicialización del puerto serie y la conexión a la base de datos, se han realizado correctamente.

2. Un nodo vecino con dirección 0013A20041255DB2 solicita datos de conexión al coordinador, y éste responde enviándolos.

3. El nodo 0013A20041255DB2 confirma que quiere unirse a la red a través del coordinador. A continuación, el coordinador valida al nodo y le envía la confirmación para que empiece a transmitir. La conclusión exitosa de este procedimiento implica la unión del nodo a la red.

4. El nodo remoto 0013A20041255DAA solicita unirse a la red a través del nodo 0013A20041255DB2. El coordinador válida al nodo y le envía la confirmación mediante un mensaje enrutado para que comience a transmitir.

5. El nodo 0013A20041255DB2 envía los datos leídos por el sensor DHT11.

6. El nodo remoto 0013A20041255DAA envía los datos leídos con el sensor DHT11.

Para una mayor simplicidad a la hora de verificar el funcionamiento del sistema, se ha optado por verificar el funcionamiento del *frontend* y realizar las comprobaciones del sistema a través de él, en vez de hacerlo a través del log del coordinador o de la consola de administración de MongoDB.

Para ello, se ha inicializado el sistema añadiendo algunos elementos:

- En primer lugar se ha registrado el sensor DHT11 como sensor de temperatura y como sensor de humedad.

- A continuación, también a través del *frontend*, se ha procedido a registrar los dos nodos disponibles para las pruebas, los cuales han sido montados en compartimentos estancos y conectados a baterías para poder moverlos de un lugar a otro fácilmente, tal y como se puede ver en la *Figura 6.2*:

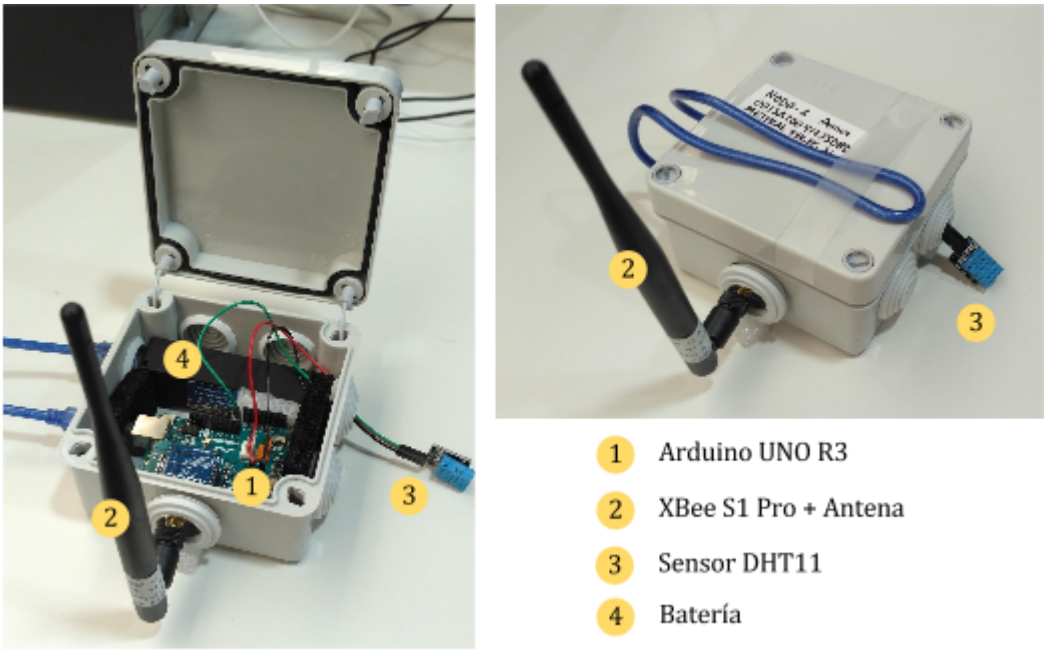


Figura 6.2: Nodo Arduino ensamblado

- Tras registrar a los sensores y a los nodos, se ha procedido a insertar datos ficticios en la base de datos a través de la consola de MongoDB y se ha comprobado si son accesibles desde el *frontend*. En la *Figura 6.3* se muestra una captura en la que aparecen tres inserciones y una consulta de la colección que alberga los datos y, a continuación, una captura de la sección de visualización de datos del *frontend* en la que aparecen dichas inserciones.

  También se ha comprobado el funcionamiento del filtrado de datos realizando búsquedas discriminantes entre las distintas entradas.

Inserción 1

Inserción 2

Inserción 3

Comprobación de la colección

```
mongo
MongoDB shell version: 3.2.9
connecting to: test
> use arsense
switched to db arsense
> db.datos.insert({idnodo:'0013A20041025DB2',idsensor:'1',valor:'22',fecha:'12-6-2016',hora:'10:35',ubicacion:'C/ Sant M
anuel 65, Mestral Telecomunicaciones S.L.',coordenadas:'13.0000,15.0000'});
WriteResult({ "nInserted" : 1 })
> db.datos.insert({idnodo:'0013A20041025DB2',idsensor:'2',valor:'20',fecha:'12-6-2016',hora:'10:36',ubicacion:'C/ Sant M
anuel 65, Mestral Telecomunicaciones S.L.',coordenadas:'13.0000,15.0000'});
WriteResult({ "nInserted" : 1 })
> db.datos.insert({idnodo:'0013A20041025DB2',idsensor:'1',valor:'19',fecha:'12-6-2016',hora:'10:37',ubicacion:'C/ Sant M
anuel 65, Mestral Telecomunicaciones S.L.',coordenadas:'13.0000,15.0000'});
WriteResult({ "nInserted" : 1 })
> db.datos.find();
{ "_id" : ObjectId("57e03eb34925bbd801db1f90"), "idnodo" : "0013A20041025DB2", "idsensor" : "1", "valor" : "22", "fecha"
 : "12-6-2016", "hora" : "10:35", "ubicacion" : "C/ Sant Manuel 65, Mestral Telecomunicaciones S.L.", "coordenadas" : "1
3.0000,15.0000" }
{ "_id" : ObjectId("57e03ed14925bbd801db1f91"), "idnodo" : "0013A20041025DB2", "idsensor" : "2", "valor" : "20", "fecha"
 : "12-6-2016", "hora" : "10:36", "ubicacion" : "C/ Sant Manuel 65, Mestral Telecomunicaciones S.L.", "coordenadas" : "1
3.0000,15.0000" }
{ "_id" : ObjectId("57e03ee24925bbd801db1f92"), "idnodo" : "0013A20041025DB2", "idsensor" : "1", "valor" : "19", "fecha"
 : "12-6-2016", "hora" : "10:37", "ubicacion" : "C/ Sant Manuel 65, Mestral Telecomunicaciones S.L.", "coordenadas" : "1
3.0000,15.0000" }
>
```

Datos recogidos por los sensores

Nodo origen	Sensor	Valor	Recepción	Localización	
0013A20041025DB2	DHT11: Sensor de temperatura	22ºC	12-6-2016 a las 10:35	C/ Sant Manuel 65, Mestral Telecomunicaciones S.L. [13.0000,15.0000]	
0013A20041025DB2	DHT11: Sensor de humedad	20%	12-6-2016 a las 10:36	C/ Sant Manuel 65, Mestral Telecomunicaciones S.L. [13.0000,15.0000]	
0013A20041025DB2	DHT11: Sensor de temperatura	19ºC	12-6-2016 a las 10:37	C/ Sant Manuel 65, Mestral Telecomunicaciones S.L. [13.0000,15.0000]	

Figura 6.3: Inserción de datos en la base de datos (superior) y visualización a través del *frontend* (inferior)

Finalmente, se ha verificado que también es posible editar, bloquear y eliminar datos, nodos, sensores, usuarios y alertas, y registrar nuevas alertas y usuarios en el sistema.

Proceso de verificación y validación con acceso a la base de datos

Una vez se ha comprobado que el *frontend* es capaz de acceder a la base de datos y representar correctamente la información que contiene y de responder a las acciones del usuario sobre las entradas de la tabla, se ha procedido a reiniciar los nodos Arduino y el coordinador de la red con el acceso a la base de datos habilitado.

Para comprobar la unión de un nodo a la red, se ha accedido a la sección de nodos y se ha comprobado en el campo “Estado en la red” si aparece como validado, tal y como puede verse en la *Figura 6.4*.

Esta comprobación permite verificar simultáneamente si un nodo puede ser correctamente enrutado, ya que como se ha dicho, el nodo 0013A20041255DAA tiene restringido el acceso directo al coordinador y, sin embargo, aparece como validado.

En la *Figura 6.5* se muestra, además, una consulta de la colección que almacena las rutas para confirmar que dicho nodo sigue la ruta esperada a través del nodo 0013A20041255DB2.

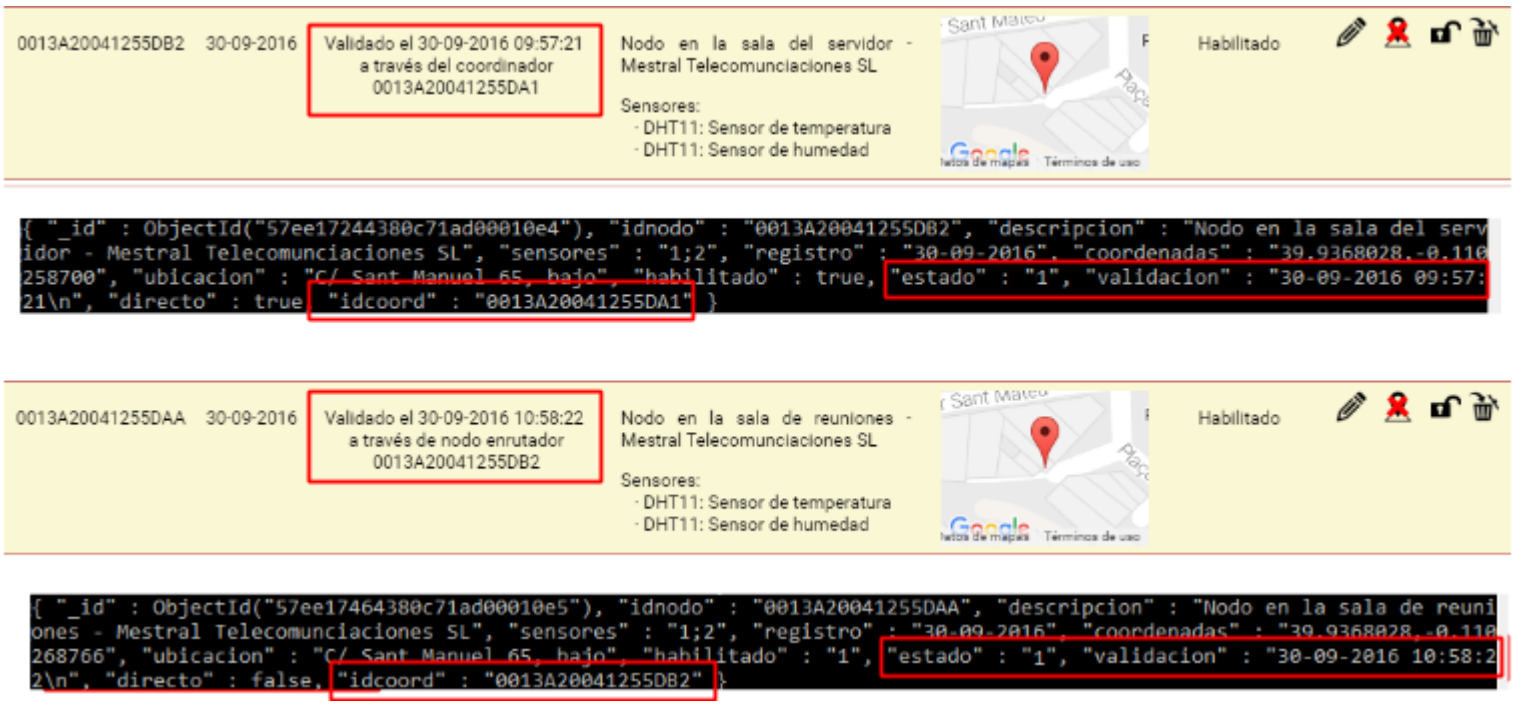


Figura 6.4: Comprobación de la validación de nodos en la red

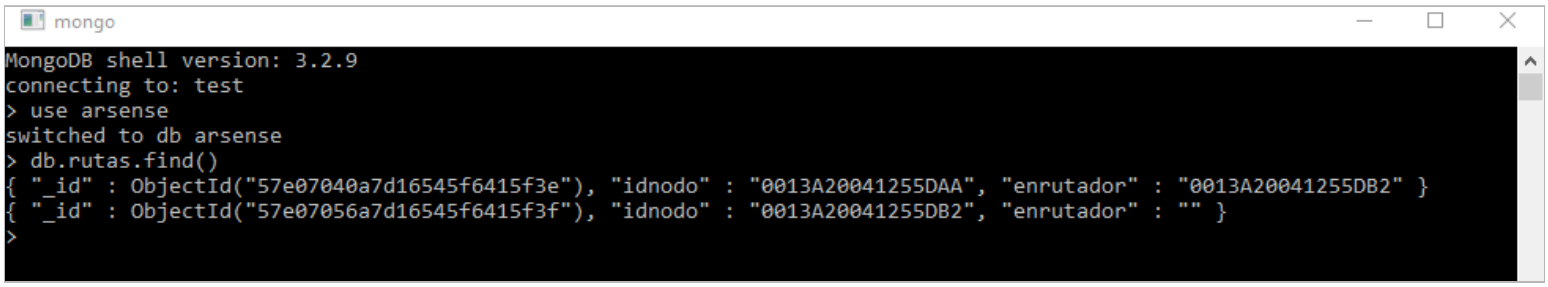


Figura 6.5: Comprobación de las rutas a través del terminal de MongoDB

Como se puede observar en la *Figura 6.5*, la entrada correspondiente al nodo con el identificador 0013A20041255DAA tiene como enrutador al nodo 0013A20041255DB2, el cual, a su vez, tiene el campo "enrutador" vacío. Esto es interpretado por el coordinador como que puede alcanzar al nodo 0013A20041255DB2 directamente, y que si quiere hacer lo mismo con el nodo 0013A20041255DAA debe hacerlo a través del primero, componiendo la ruta 0013A20041255DB2 -> 0013A20041255DAA.

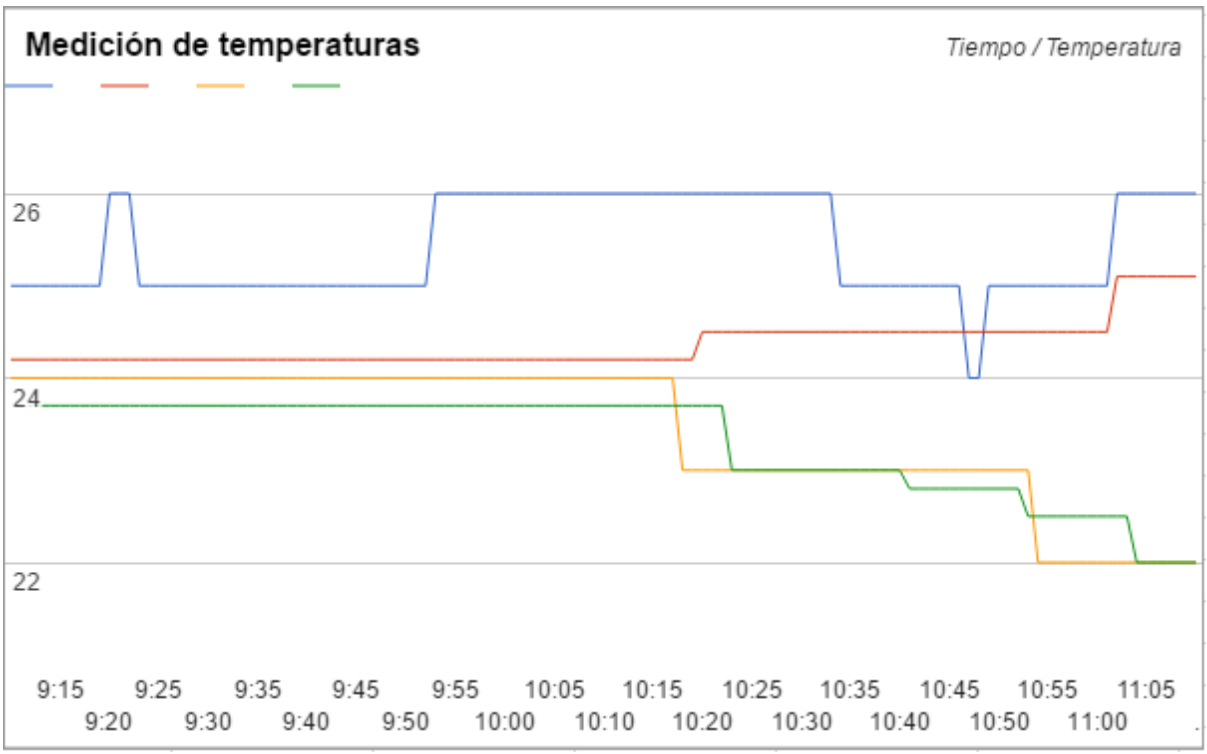
Para rutas más largas el comportamiento es el mismo, se itera de entrada en entrada hasta alcanzar a aquel nodo que es accesible directamente por el coordinador de la red.

A continuación, se pasa a verificar la recepción de datos *in situ*.

En primer lugar se realiza una prueba en el interior de la oficina. Durante dos horas, el nodo Arduino con el identificador 0013A20041255DB2 es colocado en la sala de reuniones, mientras que el otro, con identificador 0013A20041255DAA, se coloca en la sala del servidor.

Periódicamente, cada 10 minutos, mediante un termómetro manual, se realizan mediciones de las temperaturas en ambas estancias.

Tras concluir el periodo de dos horas, se realiza una comparación de las dos mediciones. Dicha comparación muestra que los datos recogidos mediante los nodos Arduino se mantienen próximos a los datos recogidos manualmente, pese a las variaciones producidas por la poca exactitud del sensor DHT11 (que, recordemos, tan sólo puede medir valores enteros con una precisión del 5% ó ±2ºC). La gráfica comparativa se muestra en la *Figura 6.6*:



Azul: nodo en la sala del servidor / Rojo: medición manual en la sala del servidor
Amarillo: nodo en la sala de reuniones / Verde: medición manual en la sala de reuniones

Figura 6.6: Gráfica comparativa entre los datos recogidos por los sensores y los anotados manualmente

Finalmente, se comprueba si el sistema es capaz de funcionar de forma estable durante largos periodos de tiempo. Para ello, se mantiene conectado durante cuatro días directamente a una toma de corriente para asegurar un suministro de energía constante.

Al transcurso de dicho periodo de tiempo, se obtienen un total de 4974 registros.

Se estudia, además, el log del coordinador para comprobar si durante dicho periodo el sistema ha sufrido alguna anomalía como, por ejemplo, un reinicio o la pérdida de conexión con alguno de los nodos.

Tras comprobar que no es así, se realiza una última prueba colocando uno de los nodos en el exterior. De este modo las condiciones de funcionamiento se asemejan más a un entorno real, con distancias algo más elevadas e interferencias atmosféricas y físicas.

Tras 48 horas se comprueba de nuevo el registro de datos y el log del coordinador para asegurarse de que no existen anomalías.

Finalmente, tras todas las comprobaciones descritas, se da por válido el funcionamiento del sistema, ya que cumple con los requisitos especificados en el proyecto: es capaz de enviar y recibir datos recogidos por los nodos a través de sus sensores y hacerlos llegar hasta la base de datos en el servidor, en donde pueden ser consultados a través del *frontend* web.

Capítulo 7

Conclusiones

La versión actual de la red de monitorización posee la capacidad de recolectar datos de sensores y enviarlos a una base de datos. Además, es lo suficientemente autónoma en el sentido técnico como para que, una vez instalado el programa en los nodos Arduino y registrado el nodo en la base de datos, éste pueda unirse por sí solo a la red y comenzar a transmitir información. Por tanto, cumple con los objetivos del proyecto planteado por la empresa.

Sin embargo, una futura revisión orientada a su comercialización debería incluir algunas mejoras, como por ejemplo:

- La posibilidad de configurar los módulos XBee o los nodos mediante comandos remotos de manera que, ante un problema, en una primera instancia no sería necesario un acceso físico a los nodos o al coordinador para reconfigurarlos o modificar algunos parámetros. En este aspecto podría aprovecharse el sistema de cabeceras implementado para crear un nuevo tipo de mensajes para transmitir comandos.

- La posibilidad de realizar actualizaciones vía OTA (Over The Air). Para implementar esta funcionalidad es necesario un hardware específico que la empresa no tenía disponible (un lector de microSD, por ejemplo), por lo que aunque fue contemplada en un inicio finalmente quedó descartada. Sin embargo, sería una excelente mejora ya que permitiría actualizar el software de los nodos y del coordinador remotamente para incluir mejoras o corregir errores sin necesidad de desplazarse hasta cada uno de ellos.

- Un sistema de enrutado más eficiente que permita identificar cuándo un nodo se encuentra saturado para evitar que sea sobrecargado por otros nodos vecinos que no pueden acceder al coordinador. Aunque este hecho debería ser aislado, es posible que ciertas limitaciones en el medio (por ejemplo un edificio) provoquen que nodos relativamente cercanos al coordinador deban enrutar a través de otro cercano que sí lo tenga al alcance. En este caso también entraría en juego una buen estudio del medio y del posicionamiento de los nodos.

- Una mejor gestión de la energía. Con un conocimiento más profundo de los Arduino, podría hacerse que estos tan sólo despertaran ante la llegada de un mensaje, permaneciendo la mayor parte del tiempo en modo de bajo consumo y, por tanto, extendiendo aún más su autonomía.

- Una fuente de alimentación solar propia. Al no disponer de este componente los elementos de la red de sensores se han alimentado con baterías. Por lo que su autonomía es muy limitada.

- La posibilidad de interactuar directamente con los nodos mediante bluetooth de bajo consumo (Bluetooth LE) y una aplicación para Android y IOS con el objetivo de poder obtener lecturas de los sensores en tiempo real.

- Cambios o adaptaciones en los Arduino, así como una carcasa adaptada para que sea más sencillo y versátil añadir o cambiar sensores (al estilo plug&play).

- Mejoras en el frontend de manera que puedan realizarse búsquedas de datos más complejas, con diferentes criterios, comparativas, etc; e incluso generar informes.

- El desarrollo de una API web para que los clientes puedan implementar su propio sitio web público de consulta de datos.

Así pues, pese a que la red de sensores, como prototipo, cumple su función, se puede observar que se trata de un proyecto inmaduro que aún debe mejorar para ser competitivo respecto a otras soluciones existentes en el mercado. Un ejemplo es Libelium (www.libelium.com), muy reconocida en el ámbito de las redes de sensores y que implementa un concepto de sistema como el que se ha planteado para este proyecto.

En lo que respecta al apartado personal, el desarrollo de esta red de monitorización ha supuesto un reto en muchos aspectos. En primer lugar la novedad de las tecnologías que he utilizado, si bien no en el aspecto innovador, ya que son muy conocidas, si en el desconocimiento previo que tenía de algunas de ellas (como los módulos XBee) o la inexperiencia con otras (Arduino y Raspberry Pi). La documentación, aunque ilustrativa, no siempre ha conseguido hacerme comprender lo suficiente como para llevar adelante algunas de las cosas que pretendía implementar, llevándome en ocasiones a un punto muerto en el que debía replantearme partes completas del proyecto. En muchas ocasiones el sentido común o lo aprendido a lo largo de la carrera me hacían considerar ciertas opciones que en un proyecto con estas características y con las limitaciones del tiempo y experiencia, no eran viables de implementar y, por tanto, debía buscar nuevas fórmulas a pesar de no tener la suficiente experiencia como para improvisar. Todo esto, sin embargo, me ha ayudado a aprender a simplificar los objetivos, a dejar de lado aquellos que lastraban el proyecto y a identificar las partes fundamentales de cada uno para conseguir completar otros más complejos. También ha supuesto un reto planificar un sistema tan complejo por la dificultad de las interacciones entre los distintos componentes y la entrada en juego del plano físico al tratar con transmisiones inalámbricas. No obstante, a pesar de estos altibajos estoy muy satisfecho por haber sido capaz de llevar a cabo un proyecto de estas características, aprendiendo durante el proceso muchas cosas relativas a Arduino, Raspberry Pi y las comunicaciones inalámbricas con XBee y adquiriendo mayor soltura en el uso de C++.

Bibliografía

[1] Gobierno de Canarias. Proyecto educativo Medusa. Características técnicas de Arduino. http://www3.gobiernodecanarias.org/medusa/ecoblog/ralvgon/files/2013/05/Caracter%C3%ADsticas-Arduino.pdf [Consulta: julio de 2016].

[2] Web oficial de Raspberry Pi. Raspberry Pi modelo 3 B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (en inglés) [Consulta: julio de 2016].

[3] Sparkfun. Manual de XBee S1 Prp 802.14.5. https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Manual.pdf (en inglés) [Consulta: julio de 2016]

[4] Estándar IEEE 802.15.4. Detalles técnicos del estándar IEEE 802.15.4. http://www.ieee802.org/15/pub/TG4.html (en inglés) [Consulta: julio de 2016].

[5] Wikipedia. XBee. https://en.wikipedia.org/wiki/XBee (en inglés) [Consulta: julio de 2016].

[6] Micropik. Datasheet del sensor DHT11. http://www.micropik.com/PDF/dht11.pdf (en inglés) [Consulta: julio de 2016].

[7] Wikipedia. MongoDB. https://es.wikipedia.org/wiki/MongoDB [Consulta: julio de 2016].

[8] BSON.org. BSON specifications. http://bsonspec.org/ (en inglés) [Consulta: julio de 2016].

[9] Universitat Jaume I - Ingeniería Informática. EI1062 Diseño de sistemas empotrados y en tiempo real. Tema 2 Proceso de diseño de sistemas empotrados. Autor: José Vicente Martí Avilés. https://aulavirtual.uji.es/pluginfile.php/3192611/mod_resource/content/2/Tema%202%20-%20Proceso%20de%20dise%C3%B1o.pdf [Consulta: julio de 2016].

[10] Blog informático. Topología de red: Malla, estrella, árbol, bus y anillo. http://www.bloginformatico.com/topologia-de-red.php [Consulta: julio de 2016].

[11] Digi. Knowledge Base. Escaped Characters and API Mode 2. http://knowledge.digi.com/articles/Knowledge_Base_Article/Escaped-Characters-and-API-Mode-2 (en inglés) [Consulta: julio de 2016].

[12] Digi. Knowledge Base. Calculating the Checksum of an API Packet. http://knowledge.digi.com/articles/Knowledge_Base_Article/Calculating-the-Checksum-of-an-API-Packet (en inglés) [Consulta: julio de 2016].

[13] Blogspot: Donaldmorrisey. Arduino, XBee and embedded development. Sleeping Arduino - Part 5 Wake Up Via The Watchdog Timer. http://donalmorrissey.blogspot.com.es/2010/04/sleeping-arduino-part-5-wake-up-via.html (en inglés) [Consulta: julio de 2016].

[14] Wikipedia: Advanced Encryption Standard (AES). https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (en inglés) [Consulta: julio de 2016]